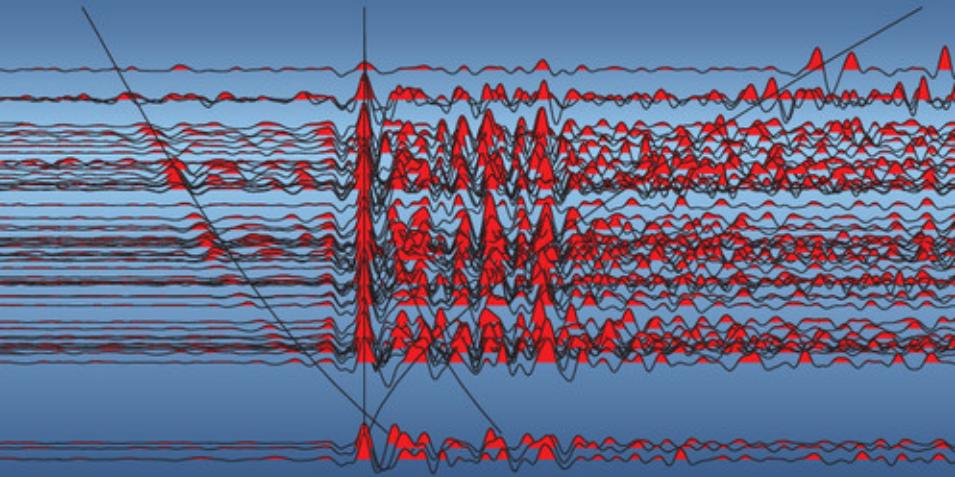
The Seismic Analysis Code

A Primer and User's Guide

George Helffrich, James Wookey and Ian Bastow



CAMBRIDGE

The Seismic Analysis Code

A Primer and User's Guide

The Seismic Analysis Code (SAC) is one of the most widely used analysis packages for regional and teleseismic seismic data. Now, for the first time, this book provides users at both introductory and advanced levels with a complete guide to SAC, enabling all users to make best use of this powerful tool.

The book leads new users of SAC through the steps of learning basic commands, describes the SAC processing philosophy and presents its macro language in full. The text is supported throughout with example inputs and outputs from SAC. For the more experienced and ambitious practitioners of the code, it also describes SAC's many hidden features, including advanced aspects of its graphics, its file structure, how to write independent programs to access and create files, and use of the methods SAC provides to integrate external processing steps into production-type data analysis schemes. Tutorial exercises in the book engage users with their newly acquired skills, providing data and code to implement the standard methods of teleseismic shear-wave splitting and receiver function analysis.

Methodical and authoritative, this combined introduction and advanced tutorial guide is a key resource for researchers and graduate students in global seismology, earthquake seismology and geophysics.

GEORGE HELFFRICH is a professor of seismology in the School of Earth Sciences at the University of Bristol. His research interests include using observational seismology to study features of the crust, mantle and core. Recently, he has based his analysis techniques on large-scale seismic array data, using SAC as the primary seismological data analysis tool. Before embarking on his research career, Professor Helffrich was a programmer who developed and supported mainframe operating systems. Bringing this experience to the seismological realm, he has contributed to the development of SAC for over 20 years.

JAMES WOOKEY is a research fellow and lecturer in the School of Earth Sciences at the University of Bristol. His research focuses on observational seismology, particularly seismic anisotropy, applied to problems from the inner core to oil reservoirs, with a recent focus on Earth's core–mantle boundary region. Dr. Wookey has spent much of his research career developing and applying novel methods for analyzing seismic data, and comparing them with predictions from mineral physics and geodynamics to better understand Earth processes. His experience with SAC spans 15 years, including as contributor to its development.

IAN BASTOW is a lecturer in seismology in the Department of Earth Science and Engineering at Imperial College, London. His research focuses primarily on the analysis of broadband seismological data from networks of temporary seismograph stations to better understand the Earth's crust and mantle. Dr. Bastow has worked extensively on tectonic problems concerning the seismically and volcanically active East African rift system, as well as on the development of Laurentia, the Precambrian core of North America. He has been a user of SAC for over a decade.

THE SEISMIC ANALYSIS CODE

A PRIMER AND USER'S GUIDE

GEORGE HELFFRICH
School of Earth Sciences, University of Bristol, United Kingdom

JAMES WOOKEY
School of Earth Sciences, University of Bristol, United Kingdom

IAN BASTOW
Department of Earth Science and Engineering, Imperial College London, United Kingdom

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

Published in the United States of America by Cambridge University Press, New York

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107045453

First published 2013

Printed in the United Kingdom by TJ International Ltd. Padstow Cornwall

Internal book layout follows a design by G. K. Vallis

A catalog record for this publication is available from the British Library

ISBN 978-1-107-04545-3 Hardback
ISBN 978-1-107-61319-5 Paperback

Additional resources for this publication at www.cambridge.org/helffrich

Contents

Preface

One of the most widely used analysis packages for regional and teleseismic seismic data is SAC (the Seismic Analysis Code). It was developed in the 1980s by nuclear test monitoring agencies in the United States, who freely made the source code and paper documentation available to academic users. From this initial distribution, the analysis package became popular in academic circles due to its ease of use and suitability for research data analysis in seismology and geophysics.

SAC's documentation was on ring-bound paper shipped along with the nine-track half-inch source code tapes that you received in the post. The academics in receipt of them generally made copies from the master document and distributed them to colleagues and students. They also tutored new users on the use of SAC, guiding them through their first session and then left them to the documentation. Consequently, much of the knowledge of SAC was passed tutorially from an experienced user, not unlike trade apprenticeship. Those intrepid enough to find and read the documentation usually exceeded their tutors' ability. The far fewer who delved into the source code learned of undocumented features of great utility. The usual reasons for this puzzling knowledge gap apply: in software development, documentation always lags feature development and, for SAC, new releases were sporadic and focused on new capability.

The original SAC documentation consisted of: (1) tutorial guide; (2) command table; (3) detailed command descriptions; (4) SAC file structure internals; (5) auxiliary program guide for programs to turn graphics to hard-copy form. Only the first of these was of any help to the new SAC user. Moreover, it is only in the tutorial guide that SAC's very powerful macro capabilities were described, but then only superficially. Consequently, the typical new user response after reading it, trying a few things out with SAC and getting confused, was to lean over to the nearest grad student and ask for help. SAC's most powerful capabilities could only be learned that way, if at all.

This book aims to remedy this continuing state of affairs when learning to use SAC. Despite the emergence of the web and web-based documentation for SAC, those available reproduce the paper ones as of about 1990. Despite some SAC development since then, no new documentation at the novice level, and little at the command description level, has been written. The main documentation effort today by the IRIS Consortium focuses on enhancing

detail in the individual command descriptions. This book is for new SAC users to lead them through the steps of learning basic commands, to describe the SAC processing philosophy and to describe the SAC macro language in full. All ideas are introduced with example input and output from SAC. For the more experienced user, the book describes the advanced features of SAC graphics: graphical interaction with traces and annotation of displays of traces with auxiliary data. We also describe the powerful, but under-appreciated feature of SAC's array data handling facilities and spectral analysis methods. Also for the experienced user, we show how to write independent programs to access and create files, and how to use the methods that SAC provides to integrate external processing steps into production-type data analysis schemes. We show this with descriptions of code and SAC macros that implement the standard methods of teleseismic shear-wave splitting and receiver function analysis. Example commands and macros for tryout and text for programs is flagged using the following scheme:

Ex-0.1

```
example text here
```

This material is linked from the publisher's web site (www.cambridge.org/helffrich) for download, keyed to the identifier in the box.

We originally developed this material to teach a three-day course in SAC data processing to new PhD students in geology and geophysics. Following the successful reception of the course, we decided to turn the lecture notes and exercises into a more permanent version that redresses the shortcomings of existing SAC documentation. We hope that this book will be of use for incoming and existing PhD students, or in undergraduate courses on seismic data processing. If it is to hand at every seismologist's desk, it will have achieved its aim.

I salute the original authors of SAC, Bill Tapley and Joe Tull, for its design simplicity and implementation clarity. There can be no better testimony to the quality of a piece of software than its being in continuous use for over 30 years.

Acknowledgements

All of us came to use SAC through some variant of my own introduction to it. My thesis advisor, Seth Stein, came by me in the computer room one day, dropped the SAC manual into my lap and said, "You might be interested in using this." At least I think it was him, but some of my fellow PhD students might have different recollections that involve their agency. I apologize for any faulty memory. In the case of IB and JW, Graham Stuart and Mike Kendall were the responsible supervisory parties.

More recently, Frank Scherbaum helped to fill in some blanks in the history of SAC's distribution. I thank him for fingering back through some dusty filing cabinets finding documents he thought he'd never need to look at again – and frankly, didn't want to.

It was Mike Kendall's suggestion that led to us teaching a course on SAC's use. His administrative duties prevented him from contributing to the book, but he deserves the blame for our increased workload and the credit for the book's inspiration when we try to trace its inception. The course went on the road, and Dave Eaton bravely funded its import and field-testing at the University of Calgary.

Finally, I thank Cheril Cheverton, the better grammarian, for reading through the first draft of the book and making countless useful suggestions despite not knowing the subject at all. That kind of criticism is the best.

George Helffrich, for the authors

Citations for development of SAC should include one or more of these articles:

Goldstein, P., and Snoke, A. (2005). SAC availability for the IRIS community. *Electronic Newsletter*, Incorporated Institutions for Seismology, Data Management Center.

Goldstein, P., Dodge, D., Firpo, M., and Minner, L. (2003). SAC2000: signal processing and analysis tools for seismologists and engineers. Invited contribution to *The IASPEI International Handbook of Earthquake and Engineering Seismology*, ed. W.H.K. Lee, H. Kanamori, P.C. Jennings, and C. Kisslinger. London: Academic Press.

CHAPTER
ONE

Introduction

1.1 WHAT IS SAC?

SAC is an acronym for the Seismic Analysis Code, a command line tool for basic operations on time series data, especially seismic data. SAC includes a graphical interface for viewing and picking waveforms. It defines a standard type of file for storing and retrieving time series and also reads files written in other data formats (SEG-Y, MSEED, GCF) used during field collection of seismic data. SAC is self-contained and does not rely on network access for any of its capabilities, including documentation, which makes it useful for field data quality control.

SAC reads data formats (CSS and GSE formats) used by nuclear test monitoring agencies. It also contains programming language constructs that provide basic methods for developing elaborate, multi-step analysis methodologies. Collectively, these features make SAC a useful interactive platform upon which customized analytical methods may be built and prototypical procedures may be developed.

SAC is widely known. The IRIS Data Management Center (DMC), one of the largest whole-Earth seismological data repositories in existence, allows data to be requested in SAC form. The instrument response information provided by the DMC's SEED reading program, *rseed*, is usable by SAC in pole-zero or *evalresp* form. Owing to SAC's longevity, a rather large and well debugged software tool ecosystem has evolved around its file format. One such tool, *jweed*, searches for and retrieves data held by the DMC. SAC data and SAC-compatible instrument response information are among its output options. This and many other programs developed by individual researchers make SAC a natural choice for time series data analysis.

1.2 HISTORY AND DEVELOPMENT

SAC initially was developed in the early 1980s by the Livermore and Los Alamos National Laboratories in the Treaty Verification Program group. The developers were led by W. C. Tapley and Joe Tull, and the package incorporated parts of Dave Harris' XAP program (Harris, 1990). SAC's Fortran source code was distributed to interested academics as it became a tried, tested and valuable system for non-commercial seismic data processing. In this early distribution epoch, there was a collegial agreement between users and maintainers to send bug fixes and improvements to the developers in exchange for use. In 1986, SAC came to GH's notice, who used it in his thesis work. By about 1990, SAC had become the *de facto* standard analysis system for academic whole-Earth seismologists worldwide.

Beginning in about 1992, development of SAC was increasingly taken over by Livermore and access to the source code became restricted through distribution agreements with the lab. This culminated in the last source code release, version 10.6f, around 2003.

During this period, SAC's Fortran code base was converted to C language using an automatic Fortran to C conversion tool called *f2c*. This was apparently done in the belief that Fortran was too restrictive a language and that it hampered further development of SAC's features. Livermore continued SAC development using the C code base with a view to future commercial sales of a seismic data analysis product called SAC2000. In this version, SAC's functionality was to be extended by adding, among other features (Vergino and Snoke, 1993), a processing log database that would record all steps in transforming a trace from a raw form into a processed form. The design allowed for the processing steps to be tentatively saved and then either committed to memory or rolled back to an earlier state. During this period, distribution of SAC's source code ceased due to ongoing development and commercial licensing restrictions.

In about 1998, the IRIS Consortium recognized that SAC's core user community, essentially IRIS' membership, had no guarantee of affordable access to SAC's source code. IRIS began negotiation with Livermore to develop two strands, one with database features for use by nuclear monitoring organizations and another without database features for academic use. The commercialization efforts focused on the database-enabled version. In 2005, IRIS took up support and development of SAC2000 without the database features, leading to a SAC version called, in this book, SAC/IRIS. There was no academic community interest in the commercial SAC release, and Livermore's support for SAC2000 was eventually withdrawn.

During this time, the Fortran 10.6d code base (later integrated with 10.6f) continued to be maintained and developed at the University of Bristol. The bugs that existed in the Fortran code base were gradually eliminated (though they continued to exist in the C source version) and SAC's functionality continued to expand, particularly in the area of array processing of interest to Bristol researchers. This is the version documented in this book.

1.3 ALTERNATIVES TO SAC

gSAC is a name inspired by the GNU Project's free versions of the C compiler (gcc) and the Fortran compiler (gfortran). gSAC was developed by Bob Herrmann at Saint Louis University in response to SAC's monolithic internal structure and its previously closed source distribution, using advances in computer platforms. Over a period of about six weeks in 2004, gSAC re-implemented SAC's basic seismic trace manipulation functionality from scratch. Now gSAC is a group effort that provides documented tools for manipulating seismic traces which happen to be stored in SAC's file format.

Seismic-Handler was developed at the Seismological Observatory Gräfenberg by Klaus Stammler and several contributors. It is a software package that defines a set of waveform modification programs on a common data format and a scripting language to string together the programs to produce an analysis stream. There is an interactive graphical user interface for observatory purposes (e.g., daily routine seismicity analysis) and a command line version for scientific research. In 2008 the Deutsche Forschungsgemeinschaft (German Research Foundation) funded a project for further development of Seismic-Handler.

SEISAN is a package similar in structure to Seismic-Handler but oriented for use by observatories involved in routine seismic analysis. The system comprises a complete set of programs and a simple database for analyzing earthquakes from analog and digital data. SEISAN includes graphical user interaction facilities on waveform data to locate events, to edit events, to determine spectral parameters, seismic moment, and azimuth of arrival from three-component stations and to plot epicenters. A database search functionality exists to extract and operate on the data for particular events. Most of the programs can operate both conventionally (using a single file with many events) or on a database. SEISAN contains integrated research-type programs like coda Q, synthetic modeling and a complete system for seismic hazard calculation. The system is freely available for all non-commercial use. SEISAN was developed at the University of Bergen by Jens Havskov and Lars Ottemöller.

SU/Seismic UNIX is an open source seismic utilities package supported by the Center for Wave Phenomena at the Colorado School of Mines. The package provides an instant seismic research and processing environment for users running UNIX or UNIX-like operating systems. The package is a set of independent programs that exchange data in a common format through pipes. It has no graphical interface. The original developers were Stockwell and Cohen, though now it is an open source project with many contributors.

AH is a UNIX-inspired set of basic seismic operations (reading, filtering, decimation, etc.) on an input stream to transform data that is then delivered onto an output stream. It is based on UNIX pipes. The system was developed in the early 1990s by Dean Witte and Tom Boyd of Lamont-Doherty/Columbia University. The C language source code is still available online, although the package is no longer actively maintained.

PITSA was written by Frank Scherbaum (then at the University of Munich, now at the University of Potsdam) in the early 1990s and runs on IBM PCs and Sun workstations. It offers utilities for simple trace manipulation, like shifting or scaling of traces, adding or concatenating traces, stacking and others. Internally, PITSA uses a data format geared toward earthquake seismology, and it also reads plain ASCII and SEED files. PITSA's graphical user interaction is based on menus, utilizing dialog boxes and pop-up menus. PITSA no longer seems to be maintained, but its source code is available.

MATLAB offers a time series toolbox.

R is a free, open source system with built-in graphics oriented toward statistical analysis. It includes a time series library, and loadable libraries exist to read and write SAC files.

DIY tools. SAC's documented file structure has a useful collection of library routines usable from C, Fortran and Python that make it easy to develop personal tools for analyzing SAC time series. See Chapter 6 for further information.

1.4 SAC VARIANTS

Fortran SAC. This is SAC as implemented in Fortran source code. It was distributed by its developers up to version 10.6f. Source code for this version was also distributed in restricted form in some versions of the IASPEI Software Library circa 2003.

SAC2000. This is SAC translated from Fortran into C source code and subsequently maintained in C. Database capabilities were the principal development addition to this version and some new commands were also implemented. This version of SAC is no longer distributed.

SAC/IRIS. This is derived from SAC2000, without the database capabilities. It is actively maintained by the SAC development team under the aegis of the IRIS Consortium and is distributed by IRIS.

MacSAC (SAC/BRIS). This variant is derived from the 10.6d Fortran source code distribution and represents a superset of the capabilities of SAC/IRIS. The principal extensions relative to SAC/IRIS are in the capabilities of the macro language and significantly expanded handling capabilities for array data.

1.5 REQUIREMENTS AND INSTALLATION

SAC/IRIS is distributed through a licensing agreement with the IRIS Consortium. Distributions are available in source and binary forms from IRIS. Binary distributions are available for 32- and 64-bit Linux systems, 32- and 64-bit Macintosh systems and Solaris systems. Windows users must build from source in the Cygwin environment.

MacSAC (SAC/BRIS) is presently available in prepackaged distributions for MacOSX systems from 10.2 onward. The system automatically builds itself under MacOS, Solaris, FreeBSD and Linux systems from source releases based on 10.6d.

1.6 SCOPE OF THIS BOOK

The SAC command repertoire is vast – over 200 in all. We will not attempt to cover them all in this book. Our goal is rather to provide an introduction to SAC's basic concepts and its basic command set. Consequently, many commands are not included. An exhaustive list of commands with a keyword index of concepts derived from the command description appears in the appendix. This list, along with the help information built into SAC, will hopefully lead you to speculate on, find and use SAC's broader functionality.

The first five chapters of this book constitute basic material. The following chapters emphasize SAC features not adequately documented anywhere else. Our goal here is to supply that documentation and to show, with examples, the occasionally surprising utility of those features.

The final chapter is a description of how SAC may be used to implement some of the standard data analysis procedures used by seismologists on teleseismic data. Though the procedures are eminently serviceable as they stand, their inclusion here is to serve as examples of how SAC's capabilities may be used more effectively. They represent the distillation of over 20 years of experience with SAC and will reward study and reflection. Looking into the future, one can imagine developing procedures for analyzing station noise levels from long sequences of MSEED field data blockettes, for aspects of ambient noise study and for normal mode seismology.

Note that some capabilities provided by SAC require significant skill to use properly, particularly correcting for instrument response. Unfortunately, the subtleties of this topic are beyond the scope of a SAC-oriented text such as this one. Other problematic areas include attenuation correction and earthquake location. SAC provides features that either perform these functions or link with standard programs that do them. Again, the scope widens significantly when these topics are included, and so, with apologies, we have opted to omit them here. Sorry.

CHAPTER
TWO

The SAC data format

2.1 PHILOSOPHY AND STRUCTURE

SAC file format

A seismic data trace is a set of data points that is continuous in time but that may not have been sampled at an even rate. SAC's simple approach to seismic data deems that there is one seismic trace per file. Each file contains a header that describes the trace (also known as metadata) and a section that contains the actual data. The header occupies a fixed-length position at the beginning of each file, followed by the variable-length data section.

Header data is of mixed type: integer and real values, logical (true/false) values, categorical values (distinct properties like explosive source, nuclear source, earthquake source), or text (station code, event identification, wave arrival type). The seismic data in a trace is a sequence of real-valued numbers representing the sampled physical property.

Alphanumeric and binary forms

There are both binary and alphanumeric (character) formats for a SAC file. The binary version is a more compact format that is efficiently read and written, while the alphanumeric format is easier for user programs to read and write.

The alphanumeric data format[1] is intended for ease of reading and writing and for transfer between different machine types. In this format, the header data is organized into

[1] Word processors should not be used to work with SAC alphanumeric data files. These are not in RTF or Word format, even though they contain text. The relevant format is TXT (in DOS terminology) and any editing software used to prepare or edit an alphanumeric file should be capable of writing this file type, which is devoid of typesetting formatting information.

sections based on the type of data, with each section subdivided into lines. While not an intuitive organization, this makes it easier to read and write the header data because all of the data on each line of the file is of the same type: integer, real, etc.

In contrast, the binary SAC format is compact and efficient to read and write. In this format, the header data is either integer or real floating point numbers (in IEEE 754 standard format). SAC binary data is always stored in single precision (32-bit) IEEE 754 floating point format.

Interconversion of formats

Converting to and from SAC files of different format is easy with SAC. From alphanumeric to binary,

```
SAC> read alpha <afile>
SAC> write binary <bfile>
```

From binary to alphanumeric,

```
SAC> read binary <bfile>
SAC> write alpha <afile>
```

2.2 CONVERSION FROM OTHER DATA FORMATS

GSE files

SAC is capable of reading files in other common data exchange formats. One common format generated by AutoDRM software (Kradolfer, 1996) is the GSE format (Group of Scientific Experts) (GSETT-3, 1995), in use by the United Nations' International Monitoring System, part of the Comprehensive Nuclear Test-Ban Treaty framework. This alphanumeric format can be directly read and written by SAC. To read GSE data into SAC and rewrite it in SAC format,

```
SAC> readgse <gfile>
SAC> write <sfile>
```

and to write GSE-format data when the original form is SAC,

```
SAC> read <sfile>
SAC> writegse <gfile>
```

Note that there is a separate SAC command to read and write GSE data; the `READ` command is not used.

Many traces can be present in one GSE file. The file usually is a string of messages produced by an AutoDRM and sometimes is the text of an e-mail. `READGSE` will skip the parts of the file that do not contain trace data and read the traces contained in the file into memory. The various traces may be selected for further processing by writing them as SAC files individually.

The conversion between GSE and SAC formats is not isomorphic. Some information is lost in the conversion from GSE into SAC format. Multiple origins for an event are lost; SAC only has a single origin associated with each trace. Arrival picks in a trace in excess of 10 are also lost. Any information loss is reported by SAC to the user, however.

SEG Y, MSEED, GCF and CSS formats

Unlike the various SAC formats and the GSE format, the other formats cannot be written by SAC, only read by it.

SEG Y

The SEG Y data format is based on a standard developed by the Society of Exploration Geophysicists for storing geophysical data. The first definition of the format was in 1973 and it was documented in 1975 (Barry *et al.*, 1975). Defined when computers were larger than automobiles and when magnetic tape was the most portable recording medium, many variants emerged as different user communities modified it to versions suited to more modern storage media. The PASSCAL project defined one of these variants to handle field data gathered during passive seismic experiments. This is the version read by SAC. Briefly, it has no 3600-byte reel header, requires that the trace code header value be seismic (type code 1) and allows only one waveform per file.

To read SEG Y data, use a variant of SAC's `READ` command,

```
SAC> read segy <sfile>
```

which reads the trace from the designated SEG Y file.

MSEED

MSEED (mini-SEED) (IRIS, 2006) is a format defined by the International Federation of Digital Seismograph Networks principally for data archiving and exchange. The data format is also output by field dataloggers. Consequently, this makes SAC a useful tool for field computers for field data quality control.

To read MSEED data, use a variant of SAC's `READ` command,

```
SAC> read mseed <mfile>
```

MSEED data is commonly collected in short packets, each with a time stamp and a sequence of samples starting at that time. Consequently, the data might not be continuous across packet boundaries. Any data overlaps or gaps will be reported by SAC when read.

GCF

Another data format encountered in the field is GCF (Güralp Compressed Format)[2] . This is a raw data format output by dataloggers manufactured by Güralp Systems. To read GCF data, use a variant of SAC's `READ` command,

```
SAC> read GCF <gfile>
```

GCF data is collected in blocks of 1024 characters. Consequently, the data might not be continuous across block boundaries. As with MSEED, any data overlaps or gaps will be reported.

CSS

CSS format is a format defined by the Center for Seismic Studies and is a waveform database format used by the International Monitoring System, with separate files for trace metadata

[2] Defined in Güralp online information at `http://www.guralp.com/gcf-format/`

and trace data (Anderson *et al.*, 1990). Consequently, at least three separate files must be grouped to describe a trace. The grouped files are associated through one master waveform metadata file. The metadata files have names with the same prefix but different suffixes. Files with suffixes `.wfdisc` and `.origin` pertain to the waveform metadata and the origin information for the event, respectively. A third file, whose location is given in the `.wfdisc` file, contains the data for one or more traces. Each trace is described by one line in the `wfdisc` file, so one CSS metadata file can describe one or more traces.

To read a CSS file in SAC, provide the name of the `wfdisc` file, either with or without the suffix:

```
SAC> readcss <cssdata>
```

This implies that given a file prefix of `cssdat`, files called `cssdat.wfdisc` and `cssdat.origin` also exist. Alternatively, the full `wfdisc` file can be specified on the `READCSS` command, which is useful when pattern matching for a group of files.

Many traces can potentially be present in a CSS file collection. Using suitable `READCSS` command options, subsets of the traces may be chosen based on the channel name, station name and frequency range of the sensor. By default, `READCSS` reads all of them.

2.3 BYTE-ORDER ISSUES

Information in SAC's file header unfortunately does not include an explicit indication of the data's type of binary format. Contemporary processors organize data in memory in either most-significant byte order (big-endian; SPARC or PowerPC order) or least-significant byte order (little-endian; DEC/Intel order).

The added complexity in reading and writing data in the proper way may lead to many errors in user-written programs that read and write SAC files. One strategy to avoid byte-order problems is to read and write them in alphanumeric format. This has the additional advantage that the data is easily verified simply by viewing the file as text. Alphanumeric format is a useful intermediate form to convert data from another format not recognized by SAC to make it SAC-readable.

For programs that analyze or change data in a processing tool chain, there is more of an emphasis on performance. In this case, working with the binary format is more efficient. SAC provides a set of user-callable subroutines for easy access to header and data in SAC binary files. These routines handle the byte-order related problems automatically and can cope with past or future changes to the data format in SAC binary files (changes in the binary file format are being considered as of November 2012 and will probably occur in 2013 or 2014). The routines are provided in library form along with SAC. See Chapter 6 for further details about accessing SAC file information from programs and SAC's built-in help entry for its library of user-callable subroutines (`HELP LIBRARY`).

Unfortunately, SAC binary file byte order is not specified in any standard. Because SAC was originally developed on big-endian machines (Sun, Prime and Ridge workstations), this is the expected byte order, and any files written by SAC will be in big-endian order no matter what the underlying machine type is. Some versions of SAC (notably SAC/IRIS) will write binary files in the machine's native format. If one is encountered, SAC will report that the format is unexpected. A utility program (called `sactosac` and distributed with SAC) is available to switch to the proper byte order. See SAC's help information for its utility programs (`HELP UTILITIES`) for further usage information.

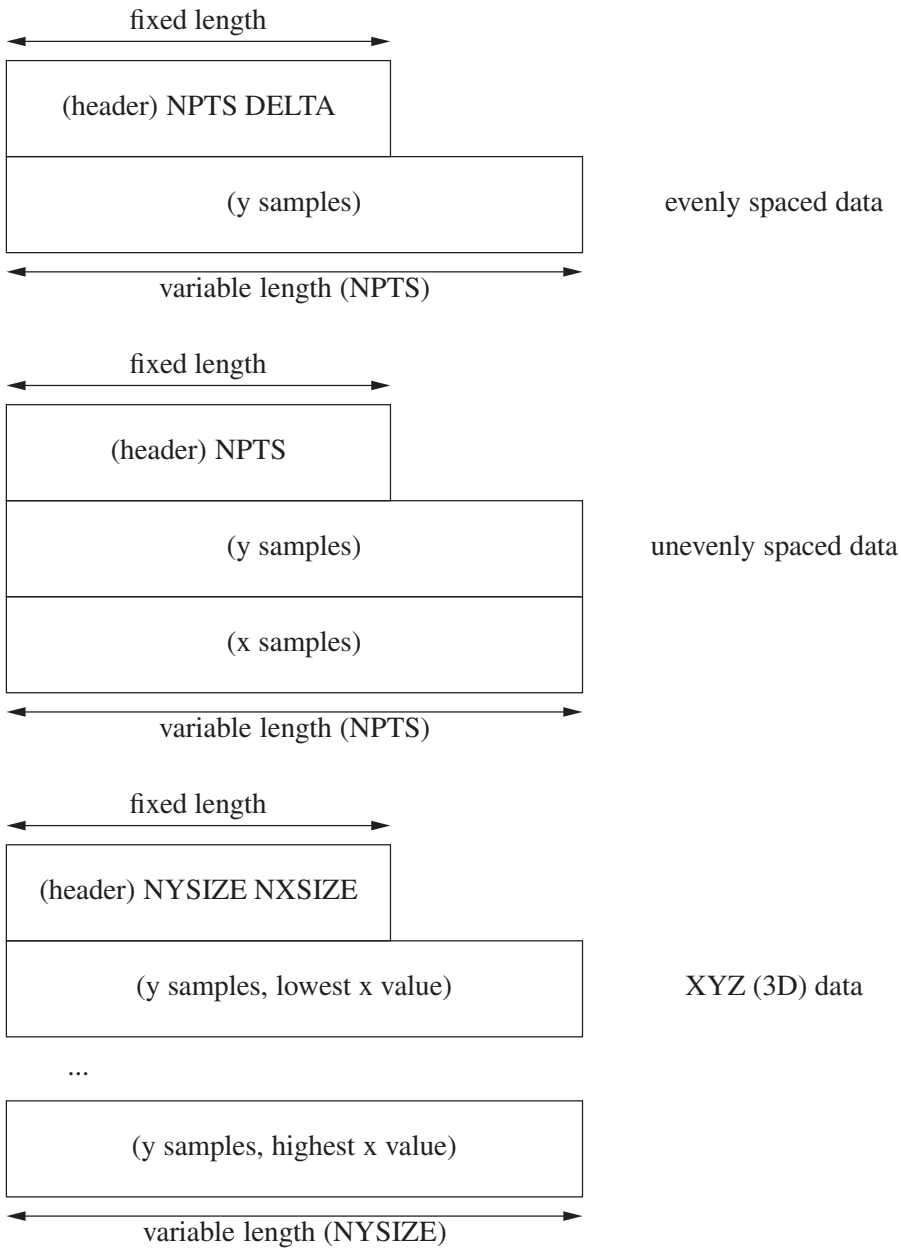


Figure 2.1 Schematic layout of SAC files of different types. All types have a fixed-length header and a variable-length data portion. For evenly spaced data, the `NPTS` entry in the file header specifies the number of data points (sampled every `DELTA` seconds). For unevenly spaced data, `NPTS` in the header specifies the number of samples, but the sampled values appear first in the data section followed by the time at which the sample was taken. For XYZ, or 3D data (see Chapter 10), the sampled values in the Y direction at the lowest X value appear first, followed by samples for the next-higher X up to `NXSIZE` X values. `NYSIZE` gives the Y dimension.

2.4 SAC FILE LAYOUT

SAC provides essentially three types of data files. The metadata-containing header is fixed length. Depending on the number of data points and the type of the file indicated in the header, a variable number of data values will follow in the file. How many data values are present depends on the type of the file. Simple time series (the most common type – IFTYPE of ITIME) have a fixed number of samples (NPTS) that are separated by a fixed sample interval (DELTA). In contrast, unevenly sampled data files (IFTYPE of IXY) have a fixed number of samples separated by random time intervals. Consequently, DELTA has no significance, and the sample values are followed by the individual times at which the samples were made. The third type of file is XYZ (or 3D data – IFTYPE of IXYZ), where samples are made on a fixed spatial grid of (X,Y) positions. In this case, the header provides the number of X and Y sample positions (NXSIZE and NYSIZE), and sample values follow for each of the NXSIZE $\times$ NYSIZE points. The ideas are sketched in Figure 2.1.

CHAPTER
THREE

The SAC processing philosophy

3.1 PHASES OF A TYPICAL ANALYSIS TASK

SAC has a complete set of reliable and well tested commands to document, view, process, transform and save data and results. The command set is confusingly large at first sight. Rather than list them all, we'll describe the steps one typically takes when processing a large seismic dataset. The commands that SAC provides relevant to each of the steps will be introduced to explain what they do and to suggest a way to remember them. The basic steps are as follows:

- Organize – standardize/prepare data for analysis.
- Interact – select a portion of the waveform to operate on.
- Process – calculate a property / estimate a parameter / transform data.
- Display – graphically show the result of processing to verify its proper functioning and to validate the data to the analyst.

Organize

Naming

The processing phase of a seismic experiment usually begins with a data delivery via hard disk, e-mail attachment, USB stick or whatnot containing data in some format. Typical formats encountered nowadays are SEED, a *tar* archive containing SAC files associated with a particular day or a particular event, or perhaps raw GCF or MSEED data straight from a field experiment. The files are rarely named appropriately. For example, a typical file group of three-component data produced by the *rseed* program (distributed by the IRIS DMC to read SEED datasets), might be:

```
2012.055.12.34.56.7777.YW.MAIO.01.BHE.Q.SAC
2012.055.12.34.50.6666.YW.MAIO.01.BHN.Q.SAC
2012.055.12.34.54.5555.YW.MAIO.01.BHZ.Q.SAC
```

representing data from a broadband field experiment recorded by the station MAIO. Rather than coping with the long file names, key them to a particular earthquake suitable for the teleseismic study in mind. For example, the event might be tersely represented by a two-digit year code, month code and day code: yymmdd. Then transform the long file names into a station name, event name and component name (BHE, BHN, BHZ) of the following form:

```
MAIO.120224.BHE
MAIO.120224.BHN
MAIO.120224.BHZ
```

This is best done using UNIX commands provided by the shell. For example, with the file names provided above, a useful set of *sh* commands to use to rename the files might be:

```
for c in BHE BHN BHZ ; do
   mv 2012.055.*MAIO*.$c.Q.SAC MAIO.120224.$c
done
```

Given SAC's command syntax, it is best to organize data with the component name at the end of the file name and the station name at front. Thus, using the *ls* command it is possible to search for file names that recorded a particular event:

```
ls *.120224.BHZ
```

or for all events recorded at a particular station:

```
ls MAIO.*.BHZ
```

Event information

After the files are properly named, standardize the data for each event. For example, it is possible to set the event information (origin time, latitude, longitude, depth and magnitude) into each file header for traces that recorded a particular earthquake.

Station and component information

Other items associated with a trace are the station location (latitude, longitude and elevation) and the orientation of the sensor. Some stations, particularly temporarily deployed ones, often do not have their N and E components pointing accurately north and east. This is important information to correct in the file header if it has not been provided automatically by the data center.

Trace length and sample rate

It might be necessary to cut the data into time windows that all have the same number of samples and have the data recorded at the same sample rate. This requires decimating or interpolating the data to the same time interval between samples and then throwing away data at the beginning or end of the trace that extend beyond the desired time window for analysis.

Discontinuous data

Sometimes the data is not continuous but is chopped up into files that contain data segments that last for a fixed time. Thus, another organizational chore might be to join the separate files into a single file representing the continuous data stream.

De-glitch data

Electronics problems with the seismometer's recording system will occasionally lead to spikes or dropouts in the seismic data stream. These so-called glitches must be edited out of the data before it is analyzed because the glitch can resemble a signal. Data glitches were more common in the era of analog data recording methods, but with digital electronics, they are becoming more rare.

Instrument response

Depending on the type of analysis, the instrument response might need to be removed in order to turn the data stream into displacement, velocity or acceleration time series. Alternatively, the instrumentation might not be homogeneous throughout a seismic network, and the analyst might want to cast the data stream as though it was recorded by a single type of instrument. This requires using the instrument response to modify the data.

Remove data mean and trend

Often a signal will have a nonzero mean or a very long-period baseline change that gives it a slope. These can thwart a signal analysis scheme and, if so, must be removed from the data before it is analyzed.

Filtering

Almost all data analysis schemes must be restricted to a range of frequencies in the data. Consequently, filtering of the time series into lowpass, highpass, or bandpass data is necessary.

Picking

Sometimes it is useful to add travel-time picks to the time series data that are predicted by some model of the process being studied. This is typically done before analysis.

Write as SAC files

Finally, the data and its metadata (event origin information, preliminary picks, etc.) is written as a SAC file. This puts the data into a homogeneous form that simplifies the following analysis.

Interact

After the data is organized, there is usually a step involving human assessment and interaction with the data. At this stage, it is useful to display the data graphically because this is less error prone than, say, typing in numbers read off a trace, and because people are visually adept and can easily evaluate a graphic. Typical interactions are as follows:

- Data quality control (QC) – discarding noisy/unsuitable traces.
- Windowing – defining a time window in the time series.
- Picking – picking a signal onset.

For these interactions, the information is presented graphically but the ultimate result is numerical or text. Data may be saved in memory for further user interaction, in the SAC file header as information associated with the trace, or in an external file. For QC steps, good file names are saved to be used as input for the next analysis step, and bad file names are removed from the analysis collection.

Process

This is the heart of an analysis task. There is usually a specialized external program, separate from SAC, to which a SAC file is passed for processing. A good program will provide text output that the user can read during the analysis for informational or verification/debugging purposes. The program should also produce output for validation of the analysis. In graphical form it is easy to confirm whether the processing was successful.

Processing might also combine a sequence of SAC analysis elements packaged to yield a useful result or to allow input to the external analysis program. For example, spectrum estimation, stacking of multiple traces, waveform picks, or trace sample arithmetic might be required.

Display

The best way to confirm that processing worked properly is through graphical output. The display should follow the processing step so that the analyst can see whether the processed signal is good. Because SAC has built-in commands to plot time series and two-dimensional data, the display step usually involves reading a file that the processing step wrote. Consequently, SAC provides a simple library of subroutines for programs to use to write files.

3.2 COMMAND SUMMARY FOR EACH PHASE

These processing steps provide a framework for introducing SAC's basic command set.

Organizing data

- Read data: `READ`, `READGSE`, `READCSS`
- Change data in file headers: `CHNHDR`
- Cut/merge data to common length: `CUT`, `CUTIM`, `MERGE`
- Put data on a common time base: `CHNHDR ALLT`, `SYNCHRONIZE`
- De-glitch: `RGLITCHES`
- Remove mean and trend: `RMEAN`, `RTREND`
- Taper: `TAPER`
- Filter operations: `LOWPASS`, `HIGHPASS`, `BANDPASS`, `BANDREJ`
- Instrument response: `TRANSFER`
- Add travel-time picks: `TRAVELTIME`
- Preview and winnow: `PLOT1`, `MESSAGE`, `SETBB` using *reply*
- Write data: `WRITE`

Interacting with the user

- Graphical interactions: `PLOTPK`, `PLOTRECORDSECTION`
- Text interactions: `MESSAGE`, `SETBB` using *reply*
- Graphical display: `PLOTC`, `XLABEL`, `YLABEL`, `TITLE`, `AXES`, `TICKS`, `COLOR`, `LINE`

Processing

- Invoke a UNIX command/external program: SYSTEMCOMMAND, $RUN ... $ENDRUN
- SAC processing facilities: APK, SPE subprocess commands, SSS subprocess commands

Display

- Graphical interactions: PLOTPK, PLOTRECORDSECTION
- Text interactions: MESSAGE, SETBB using *reply*
- Graphical display: PLOTC, XLABEL, YLABEL, TITLE, AXES, TICKS, COLOR, LINE

3.3 FURTHER INFORMATION ABOUT SAC COMMANDS

SAC itself has a built-in help facility that may be used to obtain more information about SAC commands. The most basic command is HELP COMMANDS. It provides a list of all of SAC's commands, which admittedly may be overwhelming for the new user, but can remind the experienced user of that forgotten command name.

More useful is HELP APROPOS followed by a word. SAC scans its online help for every command with the word in its description. For example, commands relating to the GSE file format may be found in this way:

```
SAC> help apropos gse

READGSE - Read data files in GSE 2.0 format from disk into memory.
WRITEGSE - Write data files in GSE 2.0 format from memory to disk.
SAC>
```

This lists two SAC commands, READGSE and WRITEGSE, that are related to GSE file handling.

Once a potentially useful command is found, more detailed help can be obtained by the HELP command. Continuing with the previous example, information about the command to read GSE files is obtained like so:

```
SAC> help readgse

 SUMMARY:
 Read data files in GSE 2.0 format from disk into memory.

 SYNTAX:
 READGSE [MORE] [VERBOSE {ON|OFF}] [SHIFT {ON|OFF}]
         [SCALE {ON|OFF}] [DIR <dir>] <file> ...

 INPUT: ...

 DESCRIPTION:
     See the READ command for general details about file reading.

 A single GSE file can contain more than one trace.  All present
 in a GSE file will be read into SAC memory.  Waveform formats of
 INT, CM6, CM7 and CM8 can be read.  The following GSE data
 messages can be handled: WAVEFORM, STATION, CHANNEL, ARRIVAL,
 ORIGIN.  If the file begins with a WIDx record, it is ...
```

The output shows SAC providing the command summary, its syntax and expected input, and then a description of what it does.

If only the command syntax is wanted, use the SYNTAX command:

```
SAC> syntax readgse

 SUMMARY:
 Read data files in GSE 2.0 format from disk into memory.

 SYNTAX:
 READGSE [MORE] [VERBOSE {ON|OFF}] [SHIFT {ON|OFF}]
         [SCALE {ON|OFF}] [DIR <dir>] <file> ...

 SAC>
```

There are also online sources of information about SAC commands. A useful one is located at the IRIS DMC, accessible through the following URL (active as of November 2012):

```
http://www.iris.edu/software/sac/commands/func_commands.html
```

CHAPTER
FOUR

Basic SAC commands

4.1 COMMAND STYLE

SAC commands are typed from the command line or read from a file. After each command is processed, SAC reads another command from its input source until it is told to stop or the input is exhausted.

Commands are single verbs (e.g., `READ`, `WRITE`), or a compound phrase (e.g., `FILTER-DESIGN`). Abbreviations exist for the longer or commonly used command names. A series of options that control the command's actions follow the command name. Command names or options may be typed in upper or lower case. However, when file names appear in commands, case does matter, and SAC preserves it.

White space separates options and the command name, and can even precede the command name. This is useful for indenting groups of commands for documentation purposes.

Multiple commands may be placed on the same line separated by the ";" (semicolon) character and will be processed left-to-right as they appear on the command line. Any command whose first character is * is a comment and is ignored. Thus the string ;* introduces a comment in the command listings that follow.

SAC supplies default command options if they are not specified. Command options, once set, stay in force for future uses of the same command. This provides a way to tailor personal command defaults. SAC can read a file of commands setting your personal defaults before it reads the input. They will be described in detail in Section 4.10.

4.2 COMMAND HISTORY

If told to, SAC will remember typed commands and allow them to be re-entered without typing them in full. The text of previous commands may be recalled and corrected or

modified and re-entered as a new command. By default, this only affects commands in the current SAC session. To recall commands across sessions, save the transcript in a file using the TRANSCRIPT command. To turn on transcription and save the commands in the file "~/.saccommands", type:

```
SAC> TRANSCRIPT HISTORY FILE ~/.saccommands
```

The file can be local to your present working directory or global. Change the file name to choose where it is placed. The previous example sets up a global transcript in a file called ".saccommands" in your home directory. The default history file location, however, is a file local to your working directory called, ".sachist".

The command history is lost if SAC stops unexpectedly (by being killed or interrupted). Only when explicitly stopped (by QUIT or finding the end of the input) is the transcript saved for future use. See HELP TRANSCRIPT for a more general listing of processing steps.

4.3 READING AND WRITING DATA

Rather unsurprisingly, the basic command to read data is READ. It is abbreviated R.

```
SAC> help read

 SUMMARY:
 Reads data from SAC, SEGY, GCF or MSEED data files on disk into
 memory.

 SYNTAX:
 READ <options> {<file>|<wild>|<url>} ...
```

READ can read one or multiple traces. It defaults to SAC binary format, but READ can read files in alphanumeric format, XML format, and other standard types (see Chapter 2). When reading new files, they replace the traces stored in memory, but this can be changed to add further traces to the collection in memory.

Files are specified by their full names or by using a wildcard to describe a pattern to apply to the file names and only reading the ones that match. The wildcards are the standard UNIX ones:

- * matches any string of characters including an empty string;
- ? matches any single (non-null) character;
- [A,B,C] matches any character (or string) in the list.

Traces are read in the order specified. File names and wildcards may contain directory prefixes.

Reading examples

Here are a few examples that show different ways to use READ. Names can be wildcarded

```
SAC> R SWAV.BH?
SWAV.BHE SWAV.BHN SWAV.BHZ
```

or listed

```
SAC> R SWAV.BHE SWAV.BHN SWAV.BHZ
```

The READ MORE option appends traces to the set in memory

```
SAC> R SWAV.BHE
SAC> R MORE SWAV.BHN
SAC> R MORE SWAV.BHZ
```

Directory references can be relative or absolute and even include wildcards

```
SAC> R /tmp/SWAV.BH?
SAC> R ../SWAV.BH?
```

Writing data

The WRITE command outputs the current traces stored in memory. It is abbreviated W.

```
SAC> help write

 SUMMARY:
 Writes data in memory to disk.

 SYNTAX:
 WRITE <options> <namingoptions>
```

The <namingoptions> represent different ways to name the files that are output from memory. They can be a list of explicit file names

```
SAC> R SWAV.BH?
SAC> W SWAV.BHE SWAV.BHN SWAV.BHZ
```

or output and overwritten using the file names they were read with

```
SAC> R SWAV.BH?
SAC> W OVER
```

or with the file names changed by appending (or prepending) a string

```
SAC> R SWAV.BH?
SAC> W APPEND .new
SWAV.BHE.new SWAV.BHN.new SWAV.BHZ.new
```

4.4 PLOTTING AND CUTTING

Devices

One of SAC's strengths is its ability to make plots on a variety of computer types and plotting devices. Devices are started (or switched to) using the BEGINDEVICES command (abbreviated BD):

```
SAC> help begindevices

 SUMMARY:
 Begins plotting to one or more graphics devices.
```

```
SYNTAX:
BEGINDEVICES [PREVIOUS]
             [[MORE] {SGF|XWINDOWS|MACWINDOWS|TERMINAL}]
```

Useful devices are Xwindows, SGF and Mac. Xwindows is the graphics system built on the Xwindows library, available on most UNIX systems. SGF (SAC Graphics File) is a hardcopy device that saves plot descriptions to a file that can be converted to PostScript and viewed or inserted in documents. Mac is a native viewer for Apple MacOSX systems (Quartz) and may not be available on all versions of SAC. (Terminal uses the features of an old (Tektronix) type of terminal that could mix graphical output with text input and output. The *xterm* terminal emulator preserves these features through the "Tek window" feature.)

Graphical plots will not be made unless there is an open graphics device. SAC provides a way to designate a graphics device to open if and when a need for one arises. Not only is this convenient, it means less screen real estate is taken up by a window that does not yet contain any information. The SETDEVICE command does this. SETDEVICE name sets the default device to any of those available through the BEGINDEVICES command.

Windows and window placement

The XWINDOWS and MACWINDOWS graphics devices allow multiple windows to be open – up to five in the present version of SAC. This can be helpful during an analysis where two graphical displays are required. For example, one window might contain an average waveform trace from a group of stations recording a particular event, and another window might contain the trace for one station. As a quality control step, visual inspection for similarity to the average waveform might lead to selection or rejection of the station for inclusion in the analysis.

Windows have a size and a position on the graphics screen. SAC allows control of the size of a window and where it is placed before it opens. Tell SAC where to put a window by using the WINDOW command:

```
SAC> help window

 SUMMARY:
 Sets the location and shape of graphics windows.

 SYNTAX:
 WINDOW <n> [XSIZE <xlo> <xhi>] [YSIZE <ylo> <yhi>]
```

The number <n> (1–5) defines that window's properties. The XSIZE and YSIZE provide the size and location of the window. They represent the low and high X and Y coordinates in terms of a relative coordinate system where the bottom left is (0,0) and the top right is (1,1).

The first window to open is numbered 1 and opens at the behest of the BEGINDEVICES command. To switch between windows, use the BEGINWINDOW command with one of the numbered windows. Plots will keep appearing in that window until it is closed (using the ENDWINDOW command) or until the window is switched using another BEGINWINDOW. ENDWINDOW removes the window from the screen, whereas BEGINWINDOW either opens a new window or switches future graphical output to it leaving the previous window visible.

Plotting data

The basic plotting commands provided by SAC are PLOT, PLOT1 and PLOT2. SAC plots time series with the time axis horizontal and the amplitude axis vertical. The command variants differ in how data from multiple traces are plotted on the device. PLOT plots one trace per window and pauses between traces. PLOT1 (abbreviated P1) organizes all the traces in a single display that shares a common time axis and plots them all in one window. PLOT2 organizes all the traces in a single display and overlays them on a single time and amplitude axis. For example, a set of three traces from built-in datasets in SAC may be displayed in different ways by PLOT1 and PLOT2 (see Fig. 4.1):

```
SAC> datagen sub regional knb.? ;* Three traces of built-in data
knb.e knb.n knb.z
SAC> plot1
SAC> plot2
```

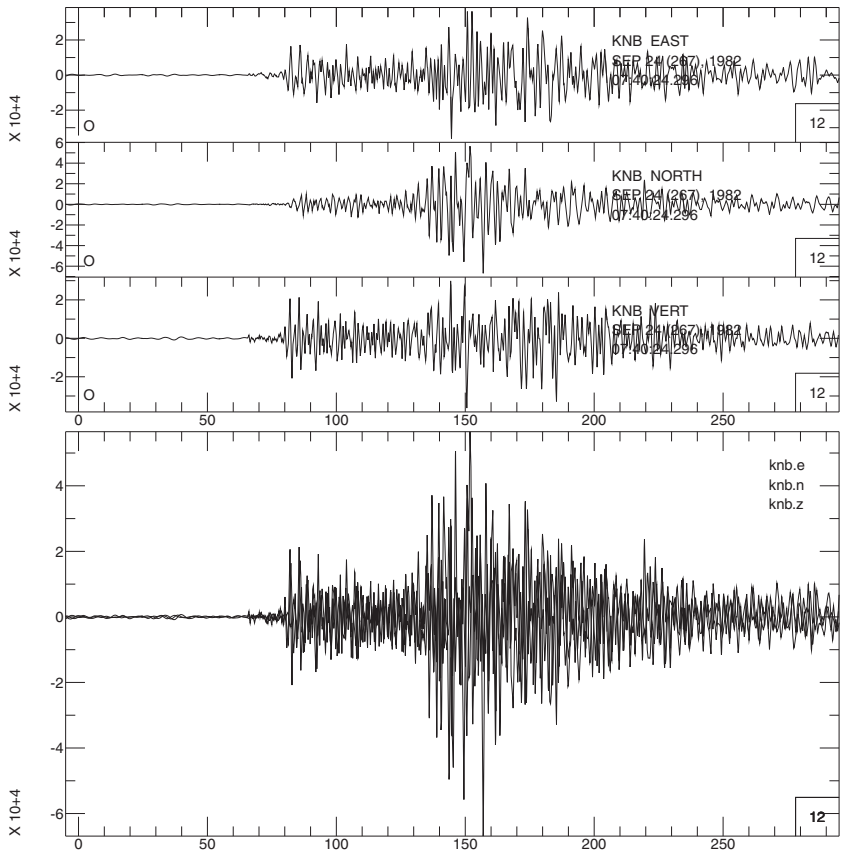


Figure 4.1 The plot made of three seismograms by PLOT1 (*top*) shows the traces in panels consisting of each trace, but sharing a common time axis. In contrast, PLOT2 (*bottom*) plots the traces overlaid in a single plot. Boxed numbers in lower right of each trace indicate a decimation factor for plotting purposes.

Many commands exist for configuring the plot size, axis labeling, plot positioning and making sub-plots in an individual display. Use `HELP APROPOS` to find further information about them.

By default, `PLOT1` plots all the traces in memory in a single display. This might yield unreadably small plots if there are many traces. Larger numbers of traces may be organized into smaller groups for display. The `perplot` option controls this. For example,

```
SAC> plot1 perplot 5
```

would limit the display to no more than five traces per page.

When plotting multiple traces in a single window, the traces might not share the same start and end times. If so, it might be useful to plot the data on a relative time scale referenced to the start time of each trace. The `absolute` and `relative` options to `PLOT1` and `PLOT2` affect this.

The `XLIM` command sets limits on the time range of the data displayed in a plot. Similarly, the `YLIM` command limits the amplitude range displayed. For fixed time limits, use

```
SAC> xlim 0 10; p1
```

The default behavior is to scale the amplitudes of individual traces to the maximum vertical space available in a plot. To make the scale the same on all traces contained in a single plot, use

```
SAC> ylim all; p1
```

To restore the default behavior, use

```
SAC> ylim off; p1
```

SAC provides controls on the style of line used to draw time series data. When plotting more than one time series on the same time axis, different line styles may be chosen to distinguish them. `LINE LIST` defines one or more line styles to be used successively in a multi-trace `PLOT2` plot. The `LINE INCREMENT` option instructs SAC to cycle through the line style list. The commands below produce the multi-trace plot in Figure 4.2.

Ex-4.1

```
SAC> datagen sub regional knb.[z,n,e]     ;* Three traces
SAC> rmean
SAC> line list 1 3 3 increment on
SAC> xlim 60 70
SAC> plot2
SAC> datagen sub regional knb.z; rmean    ;* One trace
SAC> line fill black/yellow               ;* Hymenopteral choice
SAC> plot1
```

The figure also shows another useful `LINE` option used to fill the area above and below the zero level in a trace. `LINE FILL` defines a pair of colors used to fill each trace. When

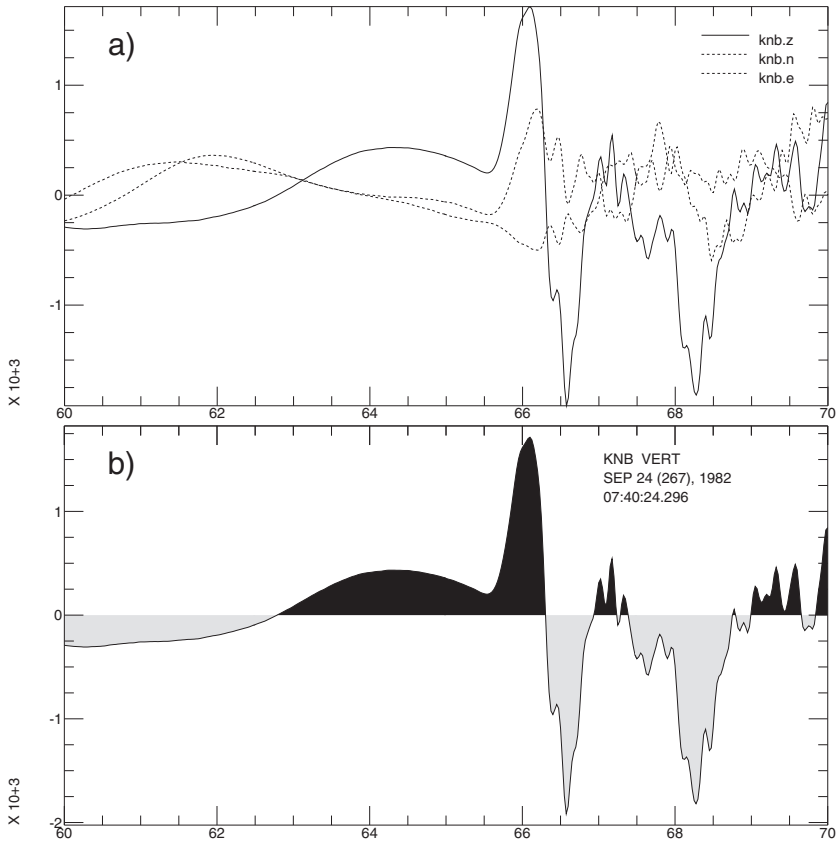


Figure 4.2 The three seismograms plotted by PLOT2 (*a*) show how different line styles enhance comprehension of the display. The line style for the horizontal component traces is different from the vertical component trace. PLOT2 shows this in the legend at top right in the display frame. Another line style feature that SAC provides to improve trace feature visibility is color-filled traces (*b*). The positive and negative areas may be filled with the same color, different colors, or limned selectively. (For color version, see Plates section.)

filling is active, it may be preferable to omit the line itself, leading to displays where trace power is the primary visual impact of the data.

There are other uses for color in plots. Trace lines may be colored. Different trace lines on a multi-trace plot may be drawn in different colors as well. The COLOR command controls these features with similar options as LINE. The basic color palette contains the colors WHITE, RED, GREEN, BLUE, YELLOW, CYAN, MAGENTA and BLACK, any of which can be selected for a line color or used in a color list. REPORT COLOR lists the color options, and Section 10.4 shows how to define your own colors.

Traces are decimated when plotted to reduce display latency (Fig. 4.1). The speed of modern computers makes a nonsense of this feature for screen displays, but it still is relevant when operating SAC remotely (as with a remote desktop on a network-connected machine) or when preserving plots (SGF files) to reduce data storage. The QDP command controls the decimation extent.

Cutting data

It is often desirable to cut information from the beginning and ending of traces to limit the data to a common time interval and to set a common time base among the traces. The CUT command enforces that only a subset of the file is read into memory. For example,

```
SAC> cut 1000 1500; r ACKN.BH?
```

would restrict the time interval from 1000 to 1500 seconds in the data. Only data in that time window is read into memory. This means that if the traces are subsequently written out, only those data will be written. So be careful – if you overwrite your original files, the data outside of the time window will be lost.

CUT and file header rewriting (the WRITEHDR command) may also interact unfavorably. SAC prevents making permanent file header changes when the header information and the data disagree. To avoid problems relating to CUT, a typical processing sequence might resemble the following:

```
SAC> r ACKN.BH?
SAC> cut 1000 1500; r          ;* Repeat previous read and cut
SAC> w prepend SHORT_
SAC> cut off; r SHORT_ACKN.BH?
```

A cut, once set, applies to all subsequent reads. Cut windows (called partial data windows by SAC) set for one file might not be appropriate for all files if, for example, a subsequent file does not contain data at the time of one of the cut limits. The CUTERR command controls what SAC should do if such errors arise. One option is to fill the missing data requested with zero. This is useful for padding data to constant length when read. It also can be handy when cutting or padding data in memory that was already read from a file when used with the CUTIM command.

SAC provides quite flexible ways to express time windows. They may be specified by times in the file or by offsets from significant markers in it. The markers can be the time of the first (B) or last (E) data point or of any pick (Section 4.5). The limit may also include an offset from the marker in seconds or number of points. Thus XLIM B N 20 represents the first 20 data points of the file, and CUT B +1 E -1 represents all but the first and last second of the file. See HELP CUT for full details. Similarly specified time windows also govern the behavior of the MTW, RGLITCHES, RMS and WIENER commands.

Permanent plots

The only use of the SGF device is for making permanent plots. When SGF is among the active devices, SAC saves a description of each plot in a file. The file, processed with one of SAC's utility programs (see HELP UTILITIES), may be converted into a form for printing or inclusion into a document for publication. The most widely usable output forms are PostScript (PS) or Encapsulated PostScript (EPS) files, though Portable Document Format (PDF) files may also be useful for document preparation.

To produce a permanent plot, activate the SGF device:

```
SAC> bd more sgf          ;* Add SGF device to active device set
SAC> p1                   ;* Make plot
SAC> bd previous          ;* Restore to previous active devices
```

Note that using the more or previous option with the BEGINDEVICES command saves or restores, respectively, the active devices.

After making the plot, SAC puts a file in the current working directory called `f001.sgf`. Subsequent plots to the SGF device will produce file names with increasing numbers in sequence. It is possible to change where SGF files are stored by choosing a different file name prefix (here it is simply `f`) and to change the sequence number used as well. See the `SGF PREFIX` and `SGF NUMBER` command for further details. The default values reset when SAC is rerun.

To translate the SGF files to PostScript, use the `sgftops` command (this is a `UNIX` command, note):

```
sgftops f001.sgf myplot.ps
```

Similarly, use `sgftoeps` to convert to EPS, and use `sgftopdf` to convert to PDF.

4.5 PICKING

Seismologists commonly use travel-time picks to mark events in a time series for further analysis. This might be the arrival time of a seismic wave, the peak amplitude in a waveform, or the begin and end time of a time window containing a signal of interest. SAC provides ways to make and save time picks by putting them into the SAC file header. The best way to make a pick is graphically, by pointing at the trace with a mouse and clicking to make the pick. The command provided for this is `PLOTPK`. Its function is to produce a plot of the trace and to start a graphical interaction with the user. SAC shows the trace with the current `XLIM` and `YLIM` options in force and displays a crosshair tied to the mouse. A keyboard character defines a pick at the current crosshair position or instructs SAC to magnify the trace (or collapse it back to its prior magnification). The most basic use of `PLOTPK` is to type a character on the keyboard to save the current crosshair position. The character identifies the name of the pick to associate with the time in the trace. The most commonly used pick names are `A`, `F` and `T0` ... `T9`. See Table 4.1 for a complete list of pick names and `PLOTPK` functions.

The most commonly used keyboard responses are:

- q – quit;
- x – pick first or second X (time) limit for a zoom window;
- o – unzoom to previous window;
- a – set `A` pick value;
- f – set `F` pick value.

Picks are stored in the file headers *in memory* and must be written out to be permanently associated with the files. By default, a pick is only associated with the trace in the crosshairs. To associate a time pick with all visible traces, use the `PLOTPK MARKALL` option. As with multiple plots, the `PLOTPK PERPLOT` option may be used to restrict the number of traces simultaneously visible to prevent excessive vertical shrinkage.

4.6 THE FILE HEADER

The file header stores metadata about the trace. These relate to:

- the trace itself (number of points `NPTS`, sample interval `DELTA`);
- the reference time of the trace (year `NZYEAR`, Julian day `NZJDAY`, ...) and the trace's relation to it (begin time `B`, end time `E`);

Table 4.1 PLOTPK cursor options

Char.	Description
K	Finish PPK
Q	Finish PPK
N	Go to next trace of multiple trace display
B	Go to previous trace of multiple trace display
X	Define bound of new time window (left first, then right)
O	Return to last time window (memory of the five most recent windows is kept)
L	List time and amplitude of current cursor position
A	Define A pick
Tn	Define Tn pick (n is a digit 0–9)
D	Define first motion DOWN
U	Define first motion UP
I	Define first motion quality impulsive
E	Define first motion quality emergent
+	Define first motion slightly UP
-	Define first motion slightly DOWN
	(Blank) define first motion unknown
@	Cancel pick information accumulated so far
P	Define P-wave arrival time
S	Define S-wave arrival time
F	Define coda length
J	Define noise level
Z	Define zero level
C	Characterize pick quality of first arrival: [I|E][U|+|-|D][0-3] uses C1, C6 from APK (C1 is weight for previous difference, C6 is weight for previous mean absolute value); set background level to 1.6 times mean absolute value amplitude. Motion sense determined if excursion is > 4× background level or if absolute value of derivative changes in a consistent sense for > 3 points. Pick quality determined by height of first three peaks relative to the background level (P1, P2, P3 here) and derivative relative to background (here K) at pick (P or S): quality 3 if K ≤ 1/2; quality 2 if K > 1/2 and P1>2; quality 1 if K > 1/2 and P1 > 3 and (P2 > 3 or P3 > 3); quality 0 if P1 > 4 and (P2 > 6 or P3 > 6)
G	Write pick information into HYPO file
H	Write pick information into HYPO file
W	Compute waveform (sets type W)
V	Compute waveform (sets type WAWF)
M	Compute waveform (sets type MWF) uses first five peaks after pick to determine an average decay rate per unit time and extrapolates it to zero to determine end of waveform

- the event (origin offset O, event depth in km EVDP, latitude EVLA, longitude EVLO, event ID KEVNM);
- the station (latitude STLA, longitude STLO, elevation STEL, name KSTNM, sensor specifier KHOLE);
- the ray path (azimuth AZ, back-azimuth BAZ, range in degrees GCARC, distance in km DIST).

Header information is either set manually or is automatically computed based on other header information (for example, AZ, BAZ and GCARC are calculated whenever STLA, STLO, EVLA, EVLO, are available or change, under control of LCALDA). Header items can have an UNDEF value, which represents the unset state. See HELP HEADER for a full list of header items.

Time representation

SAC references all times in a file relative to the zero time in the file, which is represented to the precision of a millisecond. Every other time in the file is expressed as an offset, in seconds, from the file's zero time (Fig. 4.3). This provides a very simple way to ensure time consistency between picks in a trace. However, the times are stored as single precision IEEE floating point numbers, which have only about six decimal significant figures, though their magnitude range may vary over $10^{\pm 38}$. Consequently, time precision degrades with distance from the file zero time. This is not a noticeable problem unless picks are made in very long data files. For example, times more than 1000 s from file zero only have a precision of ±1 s. (SAC may change its internal way of storing pick information in the near future to remedy this.)

Listing

The command LISTHDR (abbreviated LH) lists header information for the files currently in memory. By default, this lists all non-UNDEF (defined) entries for all files. The list can be

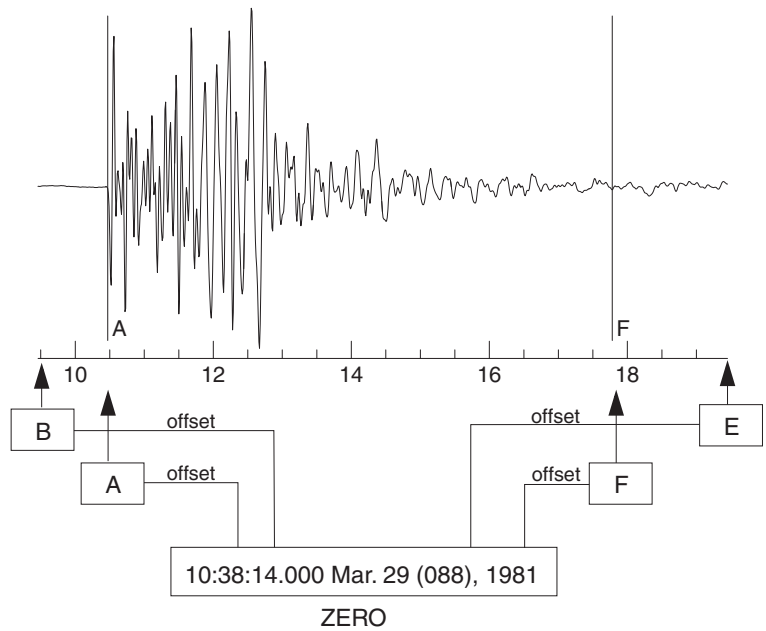


Figure 4.3 All times in a file are relative references to a time (ZERO) that has millisecond resolution. All other times (event origin time, picks (A, F), data begin B, and end E) are relative to that reference. Each time is stored in the file as an offset, in seconds, from the zero time of the file.

tailored by naming specific header variables. For example, the file zero date and time may be listed by

```
SAC> funcgen seismogram
SAC> lh kzdate kztime

  FILE: SEISMOGR
  --------------
    kzdate = MAR 29 (088), 1981
    kztime = 10:38:14.000
```

Changing

To change header information, use CHNHDR (CH).

```
SAC> help ch

 SUMMARY:
 Changes the values of selected header fields.

 SYNTAX:
 CHNHDR  [FILE <n> ...] [ALLT <v>] <field> {<value>|UNDEF} ...
```

By default, the information for all files is changed, but it may be restricted to specific files given by their order in memory. The header indicated by <*field*> is changed to <*value*>. For example, to set event information, use

```
SAC> ch evla 35.2 evlo 135.5 evdp 420.0
```

Values representing times (picks, event origin time) are values in seconds as an offset from the file ZERO time. Hence when changing a time, a real number in seconds is usually used. However, when setting an event origin time, the arithmetic is complicated by day, month and year boundaries that may appear in the trace. Time values may be specified as either GMT or as YMD times, and the arithmetic to determine the correct offset will be done automatically. For YMD times, the time is specified by seven integers: year, month, day, hour, minute, second and millisecond. For GMT times, the time is specified by six integers: year, Julian day, hour, minute, second and millisecond. (The Julian day is the day number in the year (1–365 or 1–366) and may be determined by using the -j option to the UNIX *cal* command.) Some examples might be

```
SAC> chnhdr O -45
SAC> chnhdr O YMD 2006 3 16 13 45 28 600
SAC> chnhdr O GMT 2006 075 13 45 28 600  ;* Same time as previous
```

All the times in a file header can be changed if needed for a new reference. The ALLT option does this to a specified time shift

```
SAC> chnhdr allt 415.045
```

The SAC command SYNCHRONIZE also does this, but only to the latest file start time in a group of files.

Writing

When header values change, only the copy in memory changes. The file version does not change unless the file is rewritten with `WRITE` or the updated header information is written with `WRITEHDR` (abbreviated `WH`). For example, the following commands change the file origin time and write the updated header information to the file without changing the trace data:

```
SAC> chnhdr O GMT 2006 075 13 45 28 600; wh
```

The following example sets station and event information into the header for a file and updates the header information in the file:

```
SAC> r ACKN.BHZ
SAC> chnhdr stla 64.9915 stlo -110.871
SAC> chnhdr evla 36.520 evlo 70.84 evep 183.0
SAC> chnhdr O GMT 2004 096 21 24 04 000; wh
SAC> lh O

   FILE: ACKN.BHZ
   --------------
     o = -415.0

SAC> chnhdr allt 415; wh ;* All times now relative to event origin
```

4.7 TRACE PREPARATION AND RESAMPLING

Seismic data is rarely recorded in a directly analyzable form. Instrument problems, long-period noise, temperature variations and electronic interference all imprint seismic data – especially data recorded during field deployments – with unwanted features. Data might also be recorded at different sample rates at different stations in a network, or at unnecessarily high rates that generate unprocessable amounts of data. Finally *all* data contains noise, which masks, to some extent, the signal of interest. This section is an overview of the trace preparation steps that remove many of these annoyances before data analysis. In essence, the operations remove information from the traces that lie outside the frequency range of the signal being sought.

De-glitching

SAC's `RGLITCHES` command removes spikes from data. Spikes are single samples whose values exceed the local data mean by many orders of magnitude and are usually caused by electronic noise in the recording circuitry (Fig. 4.4). Glitches are identified when data exceeds a particular absolute threshold value or differs from a running mean by more than a threshold value, or are discovered by a method that finds that a particular single sample difference is larger than the average intersample difference over a fixed time window. Glitches usually set the signal to a constant level, so a threshold method usually works best to remove the glitch. A specific time window may be set to restrict glitch removal to an interval where one is visible.

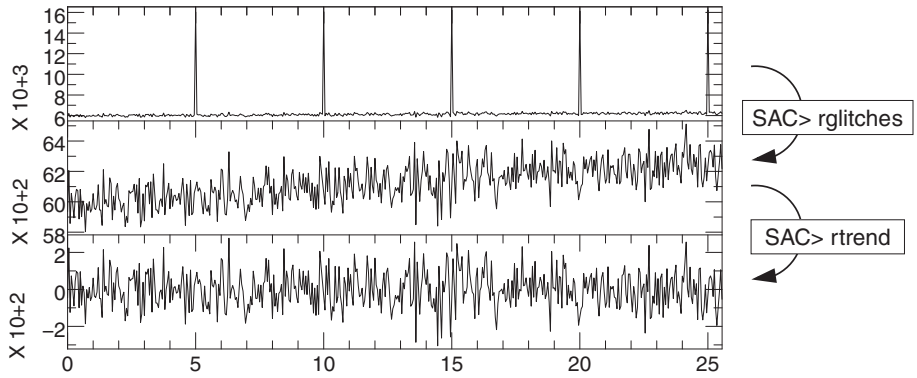


Figure 4.4 The plot made of three seismograms shows traces in successive stages of data cleanup prior to processing. (*top*) The raw data contains glitches every 5 s where the signal reaches very large values due to electronic problems. Operating on the data with the RGLITCHES command removes them. (*middle*) The repaired data contains a slope presumably due to instrument instability and has a nonzero mean. (*bottom*) After RTREND removes the trend and mean of the data, the resulting trace is centered around zero.

Mean and trend removal

Seismic data frequently displays long-period drift due to diurnal changes in ground conditions. Long-period drift can also arise when instruments are not in observatory conditions where the vault temperature is not stabilized. Expansion of the metallic parts of the instrument's casings can cause tilt, leading to a long-period signal. Thus the signal mean changes with time and can be significant during long-term drift (Fig. 4.4). Seismic signals often contain long-term trends for similar reasons.

To deal with these, SAC provides RMEAN, which simply removes the mean, and RTREND, which removes the mean and the trend and reports the slope and intercept of the best-fitting line of the data. If the TERSE option is invoked, the command sets blackboard variables (see Section 5.6) that may be used to retrieve these quantities for further processing.

Resampling

A typical data processing step is to standardize the sample rate to a value that is sensible for the intended use. There are many reasons to want to do this. File bloat reduction is one consideration, but compatibility with software-imposed sample rate restrictions in a particular analysis package is another.

Interpolation is used to attain higher sample rates and is a straightforward and easy-to-understand process. The method that SAC uses (from Wiggins, 1976) ensures continuous derivatives through the interpolated data values. The command syntax is

```
SAC> help interpolate

 SUMMARY:
 Interpolates evenly or unevenly spaced data to a new
 sampling rate.
```

```
SYNTAX:
INTERPOLATE [DELTA <d>] [EPSILON <v>] [BEGIN {<b>|OFF}]
            [NPTS {<n>|OFF}]
```

which shows that the new (higher) sample rate may be chosen through a lower INTERPOLATE DELTA value. INTERPOLATE is the command to use to convert unevenly spaced files (X–Y data pairs) into evenly spaced files, upon which spectral analysis may be done.

Obtaining lower sample rates requires more care due to aliasing of high-frequency values to lower. For example, suppose a sine wave with a frequency of 20 Hz and sampled at 40 Hz is downsampled to 1 Hz simply by using the value of every 40th sample. If the first sample chosen corresponds to a peak of the sine wave, every succeeding lower-rate sample will also be at a peak, and the trace will apparently have a nonzero mean when, on average, the sine wave mean is zero. To prevent the higher-frequency information aliasing down to lower frequencies, filtering must simultaneously be done. SAC implements such a strategy by providing finite impulse response filters to filter and simultaneously downsample through the DECIMATE command. The set of decimation factors is small (2–7), so large downsampling rates must be arranged by repeating the command. For example, 100 samples per second data could be reduced to one sample per second by decimating by 5, 5, 2 and then 2.

```
SAC> funcgen random 1 npts 1001 delta 0.01
SAC> lh delta npts b e


  FILE: RANDOM01
  --------------
    delta = 0.10E-01
     npts = 1002
        b = 0.0
        e = 10.010
SAC> decimate 5; decimate 5; decimate 2; decimate 2
SAC> lh delta npts b e


  FILE: RANDOM01
  --------------
    delta = 1.0
     npts = 11
        b = 0.0
        e = 10.0
```

4.8 ROTATION

Sometimes, seismograms are not recorded in the orientation most convenient to a particular processing method, by either accident or design. Three-component sensors – in theory, at least – record the full vector motion of the ground. Motion information recorded on any three orthogonal component axes can be rotated to form any other axis orientation.

For ease of installation, seismic instruments usually have their component axes oriented up–down (Z), north–south (N) and east–west (E). Seismologically, due to the decoupling of

the SH and P-SV wavefields in spherically symmetric, layered media, the radial (R), tangential (T) and vertical (Z) are more useful component orientations. R and T depend on the source's position relative to the instrument. The radial direction is parallel to the ground and directed along the great circle joining the source and the instrument receiving the signal from the source. The positive direction is from the source toward the receiver. Z is radially upward and T is perpendicular to the two.

SAC's ROTATE command does component rotation only on a pair of components. The trace pairs must have the same length and have the same sample rate. If the components are horizontal (as indicated by CMPINC = 90 in their header information) they can be rotated to GCARC (or, synonymously, GCP) to isolate the SH component on the T component. If the rotation is NORMAL, the three-component set (with Z) forms a left-handed coordinate system (Z up, R away from the source, T left facing along positive R), whereas if the rotation is REVERSED the three-component set forms a right-handed coordinate system (Z up, R away from the source, T to the right facing positive R). Header information for the station (STLA, STLO) and the event (EVLA, EVLO) must also be present in order to define the great circle path geometry.

Alternatively, horizontal components may be rotated through a specified angle. The ROTATE THROUGH command does a clockwise rotation through a specified angle in degrees. This type of rotation also can be done with one horizontal component and one vertical component (as indicated by CMPINC = 0 in the file header).

Three-dimensional rotations may be performed by chaining together successive ROTATE commands.

4.9 FREQUENCY-DOMAIN OPERATIONS AND FILTERING

A fundamental representation of a time series is a reduction to component frequencies that are summed together with appropriate amplitudes and phase lags to reconstruct it. The frequency coefficients constitute the time series' spectral amplitude or, more simply, its spectrum. Conversion from the time domain to the frequency domain is exact and is done using the digital Fourier transform (DFT).

SAC's FFT command does the conversion to the frequency domain and IFFT does the conversion back to the time domain. The frequency-domain representation of the signal may be examined graphically using the PLOTSP command, which plots the spectral amplitude, phase, or both. Tapering might be necessary to reduce spectral artifacts by applying a window to the data. The TAPER command provides this facility.

PLOTSP, by default, plots the amplitude spectrum and then the phase spectrum as separate plots much like the PLOT command does. Usually, the spectral amplitude is more diagnostic for analysis, and with the PLOTSP AM command, only the amplitude spectrum will be shown. For example,

```
SAC> funcgen seismogram
SAC> fft
SAC> plotsp am linlin
```

produces the (single) plot shown in Figure 4.5 on a linear scale for both the amplitude and frequency axes. Log–linear or log–log plots may be requested to show particular spectral trends.

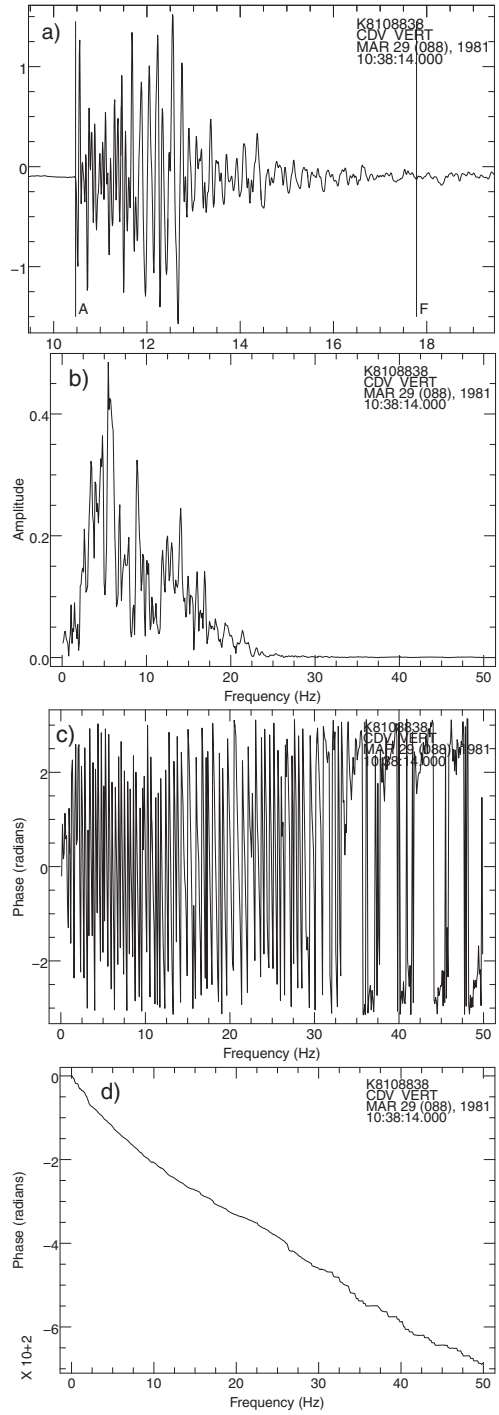


Figure 4.5 A time series (*a*), when transformed to the frequency domain by a fast Fourier transform, is represented by its amplitude spectrum (*b*) and phase spectrum (*c,d*). The amplitude spectrum, shown here on a linear scale, shows power peaked at 6 Hz. The raw phase spectrum (*c*) is not particularly useful. However, the unwrapped phase spectrum (*d*) shows a generally decreasing phase delay indicative of a causal signal.

The phase spectrum is usually uninterpretable viewed graphically because it varies rapidly between its limits of $\pm\pi$. If the phase is unwrapped in an unambiguous way it has more graphical significance. Thus SAC provides a combined fast Fourier transform (FFT) and phase unwrapping function through the `UNWRAP` command. This uses a technique to solve for the phase ambiguity to yield a continuous phase that is more helpful when assessing signal characteristics.

Once in its spectral representation, the trace may be saved and read using the normal SAC `READ` and `WRITE` commands. However, the spectral representation in the file is not the same as the time series representation. Often one wants to treat the spectral amplitude or unwrapped phase as a time series (in frequency) in its own right for further analysis (e.g., cepstral analysis, departure from linear phase). While this is not possible on the spectral representation, the spectra may be written as a time series and then re-read for further processing. The `WRITESP` command is useful for this purpose. The files that it writes (either the spectral amplitude, phase, or both) are formatted just as time series data, and if read as such by SAC are analyzable using SAC's command repertoire. To combine the spectra-as-time-series representations back into a true spectral representation, use the `READSP` command.

Filtering

If, using spectral analysis or some model of the expected signal, we can determine which frequencies present in our seismogram represent noise and which represent signal, we can filter out unwanted frequencies to improve our observation of the signal of interest. Filtering covers a very broad range of theory and methodology and cannot be covered even in an introductory form here. A recommended reference is Hamming (1989).

SAC provides filters to remove noise at low frequency, high frequency, in an intermediate band of frequencies or simultaneously at low and high frequencies. These are respectively called *highpass*, *lowpass*, *bandreject* and *bandpass* filters. Each of these has a particular shape in the frequency domain and may be visualized as a function that goes from one in some frequency range (the pass band) to zero in another frequency range (the stop band). The transition from the pass band to the stop band, conventionally chosen as the "3 db point," is where the signal drops to $10^{-3/20}$ of its value in the pass band (about 70% of the pass band value). This is usually referred to as the filter's "corner." Thus lowpass and highpass filters have a single corner frequency, while bandpass and bandreject filters require specification of two.

Another filter characteristic is the shape of the filter in the frequency domain. Filters that drop abruptly at the corner frequency sharply divide the signal portion from the noise. However, they introduce time-domain artifacts into the trace, distorting the signal. Thus smooth filter transitions are preferred to preserve signal fidelity. SAC provides four types of filters of increasingly aggressive shapes to separate signal from noise. They are Butterworth (`BUTTER`), `BESSEL` and two Chebyshev types, `C1` and `C2`.

There are further controls upon the steepness of the pass band to stop band transition for each shape. For mathematical reasons, this is called the number of poles that the filter has. The more poles, the steeper the transition and the more artifacts introduced in the time series after filtering.

The final filtering consideration is preserving the causality of the filtered signal. Increasingly aggressive filters introduce increasingly strong shifts in the phase spectrum that change the shape of a time-domain pulse to broaden it. This shifts peak positions to times

later than the peak in the corresponding unfiltered trace. To preserve peak positions, which is important for some analyses (e.g., arrival time differences), the filter is passed twice over the trace, once forward and the second time backward. The price paid is that some high-frequency energy in a pulse appears to arrive earlier than the pulse itself, destroying causality.

Thus, the filter commands that SAC provides have many options to control the features of filters. LOWPASS and HIGHPASS filters have a single corner frequency but four choices of filter type: BUTTER, BESSEL, C1 and C2. There is a choice for the number of poles, NPOLES, and a choice for the number of passes, PASSES. Finally, for the Chebyshev filters, there are further transition bandwidths (TRANBW) and attenuation (ATTEN) values to supply. The help information for BANDPASS contains a good demonstration that explains the Chebyshev controls. Owing to its simplicity, the Butterworth filter is the default filter type.

To pass through or filter out frequencies in a particular band, SAC provides the BANDPASS and BANDREJ commands. As one would expect, these require two corner frequencies to define the frequency band. Otherwise, the commands recognize the same options as do LOWPASS and HIGHPASS.

Designing filtering strategies

The compromises involved with filtering also show that a filtering strategy must be carefully thought through before applying it to data. Experimenting is extremely important, but SAC also provides help in visualizing the consequences of a filter choice through the FILTERDESIGN command, whose abbreviation is FD. The following command produces a graphical display of the Butterworth bandpass filter

```
SAC> fd bandpass corners 5 25 npoles 2 passes 2 delta 0.01
```

The resulting display (Fig. 4.6) shows the frequency-domain visualization, the phase and group delays, and a time-domain plot of the filter acting on a delta function.

Filtering decisions are complex and the consequences of a poor filtering choice might not be apparent until late in a data analysis procedure. Consequently, you should be prepared to re-create your filtered data from the raw unfiltered seismograms if you discover problems later. To make this as painless as possible, good practice is to keep the raw data in a separate directory from the analysis directory. Then filter the data and move the result to the analysis directory, preserving the raw data. SAC's macro features, described in Chapter 5, help with this process. Write a macro that takes a trace or a single group of traces from the raw data directory, filters it, and then writes the filtered data into the analysis directory. If the filter choice proves to be problematic, go back to the macro, change the filter parameters, and then re-process the data in the raw data directory. When it comes time to document and publish the results of the analysis, the macro also serves to document the precise filtering parameters used (instead of relying on memory or a lab notebook, either of which might go tragically missing).

4.10 SAC STARTUP FILES

SAC's command set has many options that define functionality. Recall that SAC remembers the previous set of options for each command and re-uses those options when the command

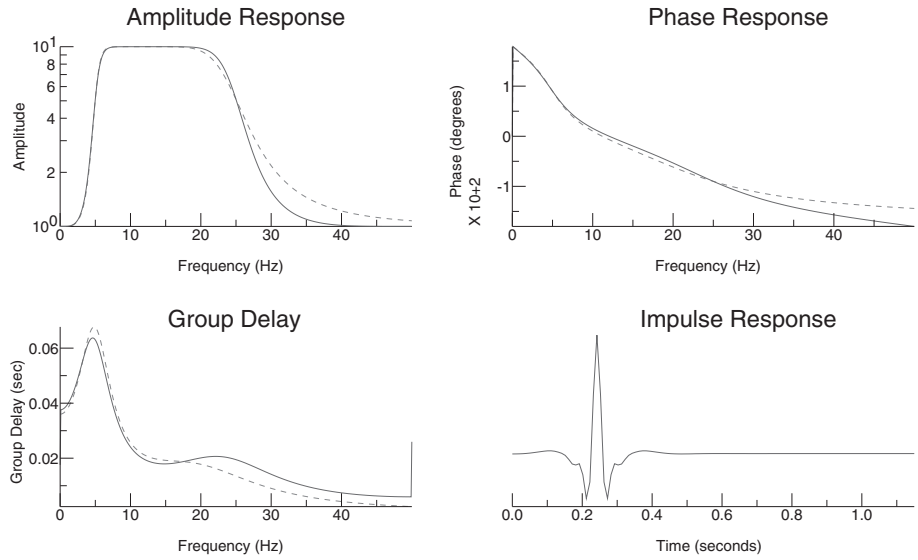


Figure 4.6 FILTERDESIGN output for a Butterworth bandpass filter with corners at 5 and 25 Hz acting on 100 Hz data. The filter is a relatively gentle two-pole filter and preserves peak position by being two-pass. However, the two passes spread the energy in the spike (bottom right) to both before and after the peak, whereas the original delta function signal was concentrated at the peak. Also note the filtered trace is negative, whereas the delta function is non-negative. The dashed line shows the analog filter prototype approximated by the digital filter (solid line).

is used the next time, unless changed. This suggests a mechanism by which a different, personally tailored set of default command options may be set up.

SAC reads commands from the standard input, defined in the UNIX system. This is usually the terminal keyboard, but, with input redirection, it can be a file. When invoked, any file name on the command line will be read by SAC before it reads from the standard input. In this way a file can be designated as a startup file to prepare your personal SAC environment. Typical commands to appear in the startup file are commands to define window sizes and locations (see WINDOW), the default graphics device (see SETDEVICE), the default macro search path (see SETMACRO), transcript location (see TRANSCRIPT), text size and style (see TSIZE and GTEXT), trace plotting resolution (see QDP), trace line style and color (see LINE and COLOR), and file header listing options (using the otherwise strange-seeming LISTHDR FILES NONE option).

A simple way to invoke SAC and direct it to read your personal startup file is to set up a shell alias to start SAC and have it read the file. If you use a Bourne-type shell (*sh* or *bash*), a way to achieve this is to put the command

```
alias sac='sac ~/.sacinit'
```

in your `~/.bashrc` or `~/.profile` file, which *sh* reads at startup. With this definition, when you type `sac`, SAC will read your commands after reading the ones in your initialization file `~/.sacinit`. A more flexible aliasing mechanism is to define a shell function as

```
func sac() { `which sac` ~/.sacinit $* ; }
```

which will let you add further files to the initial processing sequence (if you want) before reading commands from the standard input.

If you use a C-shell family shell (*csh* or *tcsh*), this will work when added to the shell startup file (`~/.cshrc` or `~/.tcshrc`):

```
alias sac ''`which sac` ~/.sacinit \!1*'
```

4.11 SAC UTILITY PROGRAMS

SAC's utilities are auxiliary programs that do useful things with SAC time series files and SAC graphics files in the UNIX environment outside of SAC proper. They are distributed in binary form along with SAC and are installed along with it in a standard installation. See HELP UTILITIES for a full list of utility programs; a few of the most useful ones will be highlighted here.

When processing SAC data files during the organization phase of a data analysis project (see Chapter 3), a shell script is often used. Frequently, processing in the script depends on properties of a SAC data file. For example, the name given to a file might be taken in part from the event name in the KEVNM header variable. The script needs access to that name, so obtaining it from the file itself would be useful.

This is the purpose of the `sachdrinfo` utility: `sachdrinfo` operates on a file (or list of file names read from the standard input) and provides, on the standard output, the values for one or more file header variables provided as command line arguments. Thus it is designed as a UNIX filter for integration with other filters in a pipe. The shell provides innumerable mechanisms to retrieve and process values in a pipe, making the utility a natural tool for use by the shell.

A research project culminates with a presentation, a publication or technical report on its results. Time series graphics might be one of the displays used. Therefore, some of SAC's utilities take a SAC graphics file (SGF) and turn it into a form useful for inclusion in the presentation. The utilities `sgftops`, `sgftoeps` and `sgftopdf` take an SGF file and transform it to each of the formats implied by the utility's name. Some commercially available software may be used to edit the result to add annotation or edit the content to improve clarity. Another of SAC's utilities, `sgftoxfig`, converts the SGF file into a form editable using the free software program *xfig*. An advantage of editing the file in this way is that the organization of the graphical elements in a plot is retained. For example, a text string that represents the title of a plot is a single, text string, possibly centered, left- or right-justified. This information is retained in the figure description, facilitating editing.

CHAPTER
FIVE

SAC macros

5.1 MACROS AND INVOKING THEM

SAC has a facility to encapsulate repeated or commonly used sequences of other SAC commands into a new command with its own name. This is somewhat like UNIX shell scripts or Windows `.bat` files. SAC calls these procedures "macros." In addition to being a convenient shorthand for repeating a series of commands, macros extend SAC's processing capabilities in ways that are not available by typing in commands. The novel feature that macros provide is a way to define local variables that may be accessed and changed and used to make decisions about which particular SAC commands are issued by the macro. This widens the SAC command language into a type of programming language.

The basic command to invoke a macro is the MACRO command (abbreviated M):

```
SAC> M TTSAC   ;* Invoke a macro in a file named TTSAC
```

The body of a SAC macro is stored in a text file. This is not a file containing word processed text in the form of Word's `.doc` or Rich Text `.rtf`. Instead, the file must be created with a program capable of writing text files (for example, BBEditLite, TextEdit, or the UNIX commands *vi*, *pico*, *nedit*). There is no particular naming convention required for SAC macros and they are not required to end with `.txt` or `.m`.

The lines inside a macro file are either SAC commands or a few macro commands that are only recognized in macros. The character "$" introduces expressions that use features of the macro language and flags those commands usable only in macros.

5.2 WRITING A SIMPLE MACRO

To write a macro called `Hello` containing the text `Hello, World!`, first enter the following text into a file named `Hello` in the current working directory.

Ex-5.1

```
MESSAGE "The macro is starting ..."
$RUN cat -
Hello, World!
$ENDRUN
MESSAGE "... and now it is finished."
```

The commands used here are a mix of SAC commands (`MESSAGE` writes a message to the user) and macro commands (`$RUN` and `$ENDRUN` run an operating system command using the text bracketed between them as input).

Next, invoke the macro in SAC using the `MACRO` command

```
SAC> m Hello
```

which results in

```
 The macro is starting ...
Hello, World!
  ... and now it is finished.

SAC>
```

5.3 TRACING MACRO OPERATIONS

In the previous example, notice that SAC did not show the macro commands it was performing, it just did them. This is useful, particularly if a macro contains a long series of commands, but makes it difficult to debug macros during development.

To see the commands inside a macro as each is performed, use the SAC command `ECHO` with either `ON` or `OFF` as an argument. When `ON`, SAC echoes each line as it is retrieved from the macro file. Each line is also echoed after macro variables (to be discussed later) are substituted into the text of the line.

The output from the same macro with `ECHO ON` would be

```
SAC> echo on               ;* Turn on macro line echoing

SAC> m Hello               ;* Invoke
 m Hello
 MESSAGE "The macro is starting ..."
 The macro is starting ...
 $RUN cat -
 Hello, World!
 $ENDRUN
```

```
Hello, World!
 MESSAGE "... and now it is finished."
 ... and now it is finished.
SAC>
```

This time, SAC writes each command as it is read from the macro (the lines shown in **bold** face) and then writes the results of executing each command.

Though not shown here, SAC echoes lines that contain macro variables with a line prefixed by ==>, after variable substitution.

The ECHO command may be restricted to any or all commands, lines read from macros, and lines after any macro processing. HELP ECHO provides more details.

5.4 SEARCHING FOR MACROS

SAC searches for files containing macros in particular places and in a particular order. Because macros are simply files, the macro name is a file name. SAC searches for a macro in sequence in the following places:

- the current directory (where SAC is running);
- directories specifically indicated by SETMACRO;
- the default, built-in macro directory.

SETMACRO is therefore a key command to direct SAC to a personal repository of macros. The following commands show two uses of SETMACRO to set up and then to augment the macro search directories:

```
SAC> SETMACRO /U/kit/sacmacro  ;* Set search path

SAC> SETMACRO MORE /tmp        ;* Add another path element

SAC> * Macro search path now is current directory,
SAC> *   /U/kit/sacmacro, /tmp, and default built-in macros.
```

Note that SETMACRO MORE appends a new directory to the macro search path rather than setting a new directory for macro search. The most common use of SETMACRO is in SAC startup files (Section 4.10) where the search path is set once and then used throughout the subsequent session with SAC.

5.5 DECISION MAKING IN MACROS

Some commands make sense only when encountered inside a macro. A command that tests a condition and then goes on to perform specific other SAC commands based on the outcome of the test is silly if you type the commands yourself: you already know the outcome of the test and can surely type the correct command! Inside a macro they make more sense because you can anticipate different work flows depending on the outcome of the test. For similar reasons, commands that loop back to repeat other commands also only make sense inside macros.

The basic conditional commands are IF / ENDIF and IF / ELSE / ENDIF. These test a condition and perform the commands in the macro bracketed by IF and ELSE or ELSE and ENDIF depending on the outcome of the test.

SAC also provides a facility to select one case from many (similar to C language's *switch* command, UNIX shell's *case* command, or Fortran's *if* command), using the construct IF / ELSEIF / ELSEIF / ELSE / ENDIF. One of the bracketed groups of commands is selected depending on the condition tested on the IF or ELSEIF command.

The SAC commands DO / ENDDO and WHILE / ENDDO provide the basic looping constructs. SAC repeatedly executes the group of bracketed commands until a condition that is tested at the beginning of each loop iteration is satisfied.

Before going into specifics about these commands, we need to introduce macro variables. These are used to build up expressions that are tested when deciding which SAC commands to run.

5.6 VARIABLES IN MACROS

SAC recognizes macro variables by special characters that delimit the variable name. The variable is recognized and then replaced with its value. After replacement, the command might be echoed (see ECHO), but then it is processed as a SAC command. Therefore, SAC variables can be used to generate part or all of a SAC command, including the name of the command.

Types and scope

Suppose there is a variable called X in a SAC macro. This may be one of three types of variable. Each has a different source for its value and is flagged by a different syntax.

- **Blackboard variables** (syntax %X%). The scope of a blackboard variable is global. A user sets a blackboard variable value by using the SETBB command. The variable exists until SAC stops running or the variable is removed using the UNSETBB command.
- **Macro variables** (syntax X). The scope of a macro variable is local to a macro. Once that macro is finished running, the variable disappears and its value is lost. Users do not set values for macro variables – they are set only by other macro commands.
- **File header variables** (syntax &n,X&). File header variables are specific to each SAC trace in memory. The syntax &n,X& refers to the value X in file number n (which is a number from 1 to the number of files in memory). Values are saved in the SAC file header (see CHNHDR) and (if the file header or data is written out) continue to exist the next time SAC is run and the file is re-read. File header variables are pre-defined; you cannot create new ones, only set or retrieve values of existing ones.

Case is unimportant in SAC macro variables. $XYZZY$ and $xyzzy$ refer to the same variable and yield the same value upon substitution. The trailing variable type flag ($, & or %) may be omitted, for convenience, if white space follows it.

Setting

Blackboard variables are the most commonly used variable in macro processing. The SETBB command sets the value (and defines the variable if it does not already exist) and the syntax

`%...%` is used to refer to variables once set. Blackboard variables are strings, but they can be interpreted as numbers when used in a suitable context. Here is an example of a string value given to a blackboard variable and used as a string and as a number in an expression:

```
SAC> SETBB x "hello"              ;* Set x to string "hello"

SAC> MESSAGE "I said %x% to you!" ;* value of x substituted

 I said hello to you!

SAC> * Now a numerical example
SAC> SETBB sum 0                  ;* Set sum to value zero
SAC> SETBB sum (%sum% + 1)        ;* increment sum
SAC> SETBB sum (%sum% + 1)

SAC> MESSAGE "sum is %sum%"       ;* value of sum substituted

 sum is 2.000000
```

The `UNSETBB` command deletes the definition of a macro variable and is used to tidy up after the processing in a macro is finished.

```
SAC> SETBB x "hello"              ;* Set x to char. string "hello"

SAC> MESSAGE "I said %x% to you!" ;* value of x substituted

 I said hello to you!

SAC> UNSETBB x                    ;* unset

SAC> MESSAGE "I said %x% to you!" ;* x forgotten now

 ERROR 1201: Could not find VARS variable blackboard X

SAC>* No longer insistent, it seems
```

5.7 EXPRESSIONS

Expressions appear in many contexts in SAC macros. One use already mentioned is to evaluate conditional macro commands. However, expressions can appear anywhere in a SAC command and may be used to create parts of it. Thus macro expressions can supply numerical values for many of SAC's commands (say, filter corner frequencies), the name of a file to be read, or even the name of the SAC command to run. The `EVALUATE` command will evaluate numerical expressions and save them to a blackboard variable. They may be used more widely to reduce reliance on temporary blackboard variables, as described below.

Syntax

A macro expression always appears inside parentheses (...). The expressions may be nested by nesting parentheses. Numerical expressions may be evaluated using the syntax, `( x + y )`, `( x - y )`, `( x * y )`, `( x / y )`, `( x ** y )`, where `x` and `y` are strings representing numerical values (** represents exponentiation). The blackboard variable examples encountered earlier showed how expressions are used to increment a blackboard variable.

Built-in functions

In addition to arithmetic operations, expressions may also be functions. The function name is the first item following the expression's opening parenthesis, and the arguments follow the name, separated by spaces. The numerical functions include square root, exponential, logarithms, etc., and the trigonometric functions. Additionally, there are miscellaneous functions that return numerical values. Table 5.1 lists these functions. Among them, *getval* is particularly useful. It will return the value of the time series in a trace at a particular time for all files in memory. If the value of only one file is wanted, say of the third file, that may be specified by using the alternative form `(GETVAL FILE 3 19.2)`.

SAC also provides a suite of functions to manipulate strings of characters. The functions to select substrings and replace text are particularly useful for transforming file names from one form into another. For example:

Ex-5.2

```
SAC> setbb str "ABC.XYZ"
SAC> message "Letters 2 and 3 of the string are (substring 2 3 %str%)"
 Letters 2 and 3 of the string are BC
SAC> message "The Jackson Five sang (change XYZ 123 %str%)"
 The Jackson Five sang ABC.123
SAC> message "Prefix is (before . %str%); suffix is (after . %str%)"
 Prefix is ABC; suffix is XYZ
SAC> message "Blank replacing the dot yields (delete . %str%)"
 Blank replacing the dot yields ABCXYZ
SAC> message "Substitution is (change . ' ' %str%)"
 Substitution is ABC XYZ
SAC> message "All in lower case: (changecase L %str%)"
 All in lower case: abc.xyz

SAC> setbb rts "ABC 123 XYZ"
SAC> message "There are (itemcount %rts%) items in the string"
 There are 3 items in the string
SAC> message "Item 2 is (item 2 %rts%)"
 Item 2 is 123
SAC> message "The last two are (items 2 3 %rts%)"
 The last two are 123 XYZ
SAC> message "Does str exist? (existbb str)"
 Does str exist? Y
SAC> message "  how about xyzzy? (existbb xyzzy)"
   how about xyzzy? N
```

Table 5.1 Built-in numerical functions usable in expressions

Command	# args[a]	Example	Description
PI	0	(PI)	Value of π
ADD	>0	(ADD 1 2)	
SUBTRACT	>0	(SUBTRACT 2 1)	
MULTIPLY	>0	(MULTIPLY 1 2 3 4)	
DIVIDE	>0	(DIVIDE 12 3)	
MINIMUM	>0	(MINIMUM 0.78 -3.5 15.2)	
MAXIMUM	>0	(MAXIMUM 0.78 -3.5 15.2)	
SQRT	1	(SQRT 2)	Square root
EXP	1	(EXP 2)	Powers of *e*
ALOG	1	(ALOG 2.5)	Natural logarithm
POWER	1	(POWER 3)	Powers of 10
ALOG10	1	(ALOG10 1000)	Base 10 logarithm
SINE	1	(SINE 0.37)	Argument in radians
ARCSINE	1	(ARCSINE 0.37)	Result in radians
ASINE	1	(ASINE 0.37)	Synonym
COSINE	1	(COSINE 0.37)	Argument in radians
ARCCOSINE	1	(ARCCOSINE 0.37)	Result in radians
ACOSINE	1	(ACOSINE 0.37)	Synonym
TANGENT	1	(TANGENT 0.37)	Argument in radians
ARCTANGENT	1	(ARCTANGENT 1.00)	Result in radians
ATANGENT	1	(ATANGENT 1.00)	Synonym
INTEGER	1	(INTEGER 1.05)	Integer part of expression
ABSOLUTE	1	(ABSOLUTE 1.05)	Absolute value
GETTIME	1,2	(GETTIME 17.05) (GETTIME MAX 17.05) (GETTIME MIN -7.05) (GETTIME MAX) (GETTIME MIN)	Time of first occurrence of value in file, or value greater than (MAX) or less than (MIN) in file, or DEPMAX or DEPMIN if no value provided
GETVAL	1,3	(GETVAL 157.32) (GETVAL FILE N 157.32)	Data values at 157.32 s in each file in memory Data value at 157.32 s in file N; N integer > 0

[a]Number of function arguments.

The *reply* function is designed for user interaction. A macro might prompt the user for a response, read it, and deliver the response for further use. In the example shown below, the response becomes part of a message:

```
SAC> setbb q "(reply 'Type in a file name: ')"
Type in a file name:  sesame
SAC> message "The file's name is %q%"
 The file's name is sesame
SAC>
```

In this example, *reply* is used to print out a prompting question ("Type in a file name:" shown in bold) that the user responds to. The response becomes the value of the *reply* function that is printed out in the message to the user.

A list of built-in character functions is given in Table 5.2. Whether blackboard or macro variables exist may be tested using the *existbb* and *existmv* functions. The function *existbb* was used earlier in one of the examples.

The function *hdrnum* is a subtle but useful built-in function used to select particular traces for processing based on values in their headers. For example, the file header has a field called KCMPNM that contains the component name. Suppose we want to find all files in memory that have the component name LE. We could do this with the following expression:

```
SAC> r /data/hinet/2005/vel/[EIHH,EDSH].L[E,N,R,T]
SAC> lh kcmpnm

  FILE: /data/hinet/2005/vel/EIHH.LE
  ----------------------------------
    kcmpnm = LE

  FILE: /data/hinet/2005/vel/EIHH.LN
  ----------------------------------
    kcmpnm = LN

  FILE: /data/hinet/2005/vel/EIHH.LR
  ----------------------------------

  FILE: /data/hinet/2005/vel/EIHH.LT
  ----------------------------------

  FILE: /data/hinet/2005/vel/EDSH.LE
  ----------------------------------
    kcmpnm = LE

  FILE: /data/hinet/2005/vel/EDSH.LN
  ----------------------------------
    kcmpnm = LN

  FILE: /data/hinet/2005/vel/EDSH.LR
  ----------------------------------

  FILE: /data/hinet/2005/vel/EDSH.LT
  ----------------------------------
SAC> setbb files "(hdrnum kcmpnm eq LE)"
SAC> message "There are (itemcount %files%) LE files."
 There are 2 LE files.
SAC> message "They are numbers %files% in memory"
 They are numbers 1 5 in memory
```

Table 5.2 Built-in character functions usable in expressions

Command	# args[a]	Example	Description
CHANGE	3	(CHANGE X Y Z)	Changes X to Y in Z
DELETE	2	(DELETE X Y)	Deletes X in Y
BEFORE	2	(BEFORE X Y)	Returns text in Y preceding X
AFTER	2	(AFTER X Y)	Returns text in Y following X
SUBSTRING	3	(SUBSTRING X Y Z)	Returns characters X to Y in Z. Index starts with 1; X or Y may be END to indicate last character
CONCATENATE	>0	(CONCATENATE X Y Z ...)	Joins all strings together
CHANGECASE	>1	(CHANGECASE D X ...)	Change case of X to upper or lower case depending on D being UPPER or LOWER
EXISTBB	1	(EXISTBB X)	Return Y or N if blackboard variable X exists or not
EXISTMV	1	(EXISTMV X)	Return Y or N if macro variable X exists or not
HDRVAL	>2	(HDRVAL H OP X ...)	Return values of file header variable H that are in relation OP to value X. Multiple OP X pairs are allowed; the logical AND of all conditions governs selection. OP may be LT, LE, EQ, GE, GT, NE, AZ or PM for numeric header variables, EQ or NE for character and symbolic header variables. Value and X compared. For logical header values, OP may be TRUE or FALSE with no X. Operators AZ and PM must always be paired, and are an azimuthal angle comparison: AZ x PM y means that H and x must differ in angle by no more than y
HDRNUM	>2	(HDRNUM H OP X ...)	As for HDRVAL, but returns number of file in memory, not the header value
ITEM	>1	(ITEM N Y Z ...)	Select item N from blank-delimited list of items Y Z ...; Y is item 1
ITEMS	>2	(ITEMS M N X Y Z ...)	Select items M through N from blank-delimited list of items X Y Z ...; X is item 1, and N may be END to indicate last item
ITEMCOUNT	>1	(ITEMCOUNT X Y Z ...)	Count of number of blank-delimited items in list
REPLY	1	(REPLY X)	Prints X and reads user reply
QUOTEBB	1	(QUOTEBB X)	Returns blackboard variable named X, preserving line breaks, for later use with WHILE READ command
QUOTEHV	2	(QUOTEHV N X)	Returns file header variable named X for file N. No macro or blackboard variable interpretation is done on the value
QUOTEMV	1	(QUOTEMV X)	Returns macro variable X. No file header or blackboard variable interpretation is done on the value

[a]Number of function arguments.

The files named .LR and .LT apparently do not have KCMPNM set in their headers. The SETBB command sets the value of the blackboard variable to the numbers of the files that contain a KCMPNM value of LE. The function *itemcount* prints the number of files by counting how many numbers are in the list. The message expands the value of files, showing the full list of file numbers.

The function *hdrval* returns the actual values in the files whose header matches, not the file number. While not so useful when testing whether a component is equal or not equal to a given value, *hdrval* is more useful when making a comparison to determine whether a particular file is within a range of values. For example, suppose we want to find the back-azimuths from the station to the earthquake that are ±50° of 10°. The following commands could be used:

Ex-5.3

```
SAC> datagen sub local *.z
SAC> setbb baz "(hdrval baz az 10 pm 50)"
SAC> setbb fnum "(hdrnum baz az 10 pm 50)"
SAC> message "Back-azimuth values %baz%"
 Back-azimuth values 53.949791 324.18115
SAC> message "Back-azimuth files %fnum%"
 Back-azimuth files 1 2
```

The result shows that two files, 1–2 in the group of nine read in, have back-azimuths in the proper range. Another example showing how range comparisons may be used with *hdrval* and *hdrnum* follows, this time with earthquake depths (the EVDP file header variable):

```
SAC> lh evdp

  FILE: /tmp/ex-1
  ---------------
    evdp = 10.0

  FILE: /tmp/ex-2
  ---------------
    evdp = 50.0

  FILE: /tmp/ex-3
  ---------------
    evdp = 250.0

  FILE: /tmp/ex-4
  ---------------
    evdp = 640.0
SAC> message "(hdrval evdp ge 200)"
 250. 640.
SAC> message "(hdrnum evdp ge 200)"
 3 4
```

Table 5.3 Status function

Command	# args[a]	Example	Description
STATUS	>1	(STATUS V)	Return SAC status item V; see below for list
STATUS items			
Name	Description		
GRAPHICS D	0 or 1 depending on whether any graphics device is open (if D is absent or ANY) or whether device D is open (D can be TERM, SGF, X, SUN, MAC)		
NFILES	Number of files currently in SAC memory		

[a]Number of function arguments.

```
SAC> * files 3 and 4 in list have evdp > 200
SAC> message "(hdrnum evdp gt 35 le 200)"
 2
SAC> * only file 2 has evdp > 35 and <= 200
```

The last group of built-in functions are those that query the internal state of SAC itself. These are useful for checking whether files are in memory or whether graphics devices are active in preparation for picking a time window in a seismogram. The *status* function provides this information. Table 5.3 lists its capabilities.

Escape character

In SAC, special characters flag the beginning of macro variables and expressions. To prevent these special characters from being interpreted as an expression, SAC provides the escape character `@`. Thus, if `@` appears before `(`, `)`, `%`, `$`, `&`, or even another `@`, the following character has its normal meaning. So, to put a % into a message, use:

```
SAC>  message "This is a 7@% solution NOT a bb var"

 This is a 7% solution NOT a bb var

SAC>  message "This is a 7% solution NOT a bb var"

 ERROR 1201: Could not find VARS variable blackboard

SAC>
```

The second message causes a parsing error because SAC expects a blackboard variable to follow the `%` character.

Evaluation order

SAC evaluates expressions in a strict sequence that may be used to embed expressions inside other expressions. Expression evaluation proceeds in three stages:

- expansion of blackboard variables %...% and macro variables $...$;
- expansion of file header variables;
- evaluation of functions.

This means that blackboard or macro variables can be inside file header variables:

```
SAC> setbb x 1                              ;* set x to 1
SAC> message "File %x% evdp is &%x%,evdp&"  ;* %x% is inside &..&
 File 1 evdp is 10.0
SAC>
```

Expressions are also nestable:

```
SAC> setbb x 1                              ;* set x to 1
SAC> message "sqrt (%x% + %x%) is (sqrt (%x% + %x%))"
 sqrt 2.000000 is 1.414214
SAC>
```

Macro variables may be nested inside blackboard variables, making it possible to implement blackboard variable arrays:

Ex-5.4

```
SAC> sc cat showme                    ;* Show the macro text
* This is the content of a macro called "showme"
setbb x1 "x1 value" x2 "x2 value"
do i list 1 2
   message "i is $i$ and x$i$ is '%x$i$%'"
enddo
SAC> macro showme                     ;* Invoke the macro
 i is 1 and x1 is 'x1 value'
 i is 2 and x2 is 'x2 value'
SAC>
```

The macro uses the DO command to set the macro variable I to the values of 1 and 2. For each value, the macro prints a message that contains the value of I and then the value of %x1% or %x2%, depending on I.

Conditions

The conditional commands IF, ELSEIF and WHILE evaluate a condition to decide which groups of SAC commands to execute. A condition is made of two expressions and a relational operator <*e1*> OP <*e2*>. The relational operator OP may be

- EQ (equal) or NE (not equal) – numerical or character expressions;
- LT (less than), LE (less than or equal), GT (greater than), GE (greater than or equal) – numerical expressions only.

(The operator names are similar to relational operators in the Fortran programming language.)

A few examples follow to show some guidelines when writing conditions.

```
if "%x%" EQ 'debug'    ;* quotes in case string contains blanks
   setbb debug 1
endif
```

Ex-5.5

```
* Below, reply is prefixed with _ to recognize no response
*   -- a blank line typed for the reply
setbb ans "_(reply 'Enter yes or no:')" ;* prompt and get answer
if "%ans%" eq "_yes"
   message "response is yes"
elseif "%ans%" eq "_no"
   message "response is no"
elseif "%ans%" eq "_"
   message "nothing typed"
else
   message "invalid response: (after _ '%ans%')"
endif
```

5.8 SUSPENSION, RESUMPTION AND ESCAPE FROM MACROS

Macros normally run until the commands in the macro file are exhausted, but a macro also may be explicitly stopped. The `$KILL` command provides a way to conditionally end a macro. For example, a (*reply* ...) expression might prompt the user to pick the beginning and the end of a waveform as follows:

```
setbb resp "(reply 'Hit return to continue or q to abort: ')"
if '_%resp%' EQ _q
   $kill
endif
```

Macros also may be temporarily suspended and resumed. While suspended, SAC returns to reading commands from the user's terminal. The user can issue any SAC command, including running new macros – these will not interfere with the macro that is suspended. This facility provides a type of co-routine structure between the macro and the command language.

The intent of the macro suspension facility is to allow errors in the processing of a macro to be corrected. For example, suppose the macro implements a procedure to rotate the components of a three-component seismogram from the Z, N and E coordinate system into the Z, R and T coordinate system. If there is no earthquake epicenter associated with the files, this rotation is impossible. The macro might detect this condition and ask the user to add event information before proceeding further. When the user adds the event information, the macro may be resumed.

The `$TERMINAL` command suspends a macro and the `$RESUME` command resumes a suspended macro. To show that there is a suspended macro running in the background, SAC changes its command prompt to show the level of suspension. The following example shows how macros are suspended and resumed:

Ex-5.6

```
SAC> sc cat susp
* This is the contents of the "susp" macro
message "Entering the susp macro ..."
$terminal
message "...continuing the susp macro"
message "Further processing would take place here"
SAC> m susp
 Entering the susp macro ...
SAC(macro001)> report xlim

    XLIM option is OFF
SAC(macro001)> $resume
 ...continuing the susp macro
 Further processing would take place here
SAC>
```

The susp macro started up and was then suspended via the $TERMINAL command. Notice that the prompt changed from SAC> to SAC(macro001)> while the macro was suspended. After processing the $RESUME command, SAC picked up where the macro left off.

This facility is useful for fixing errors. However, it requires the analyst to be highly knowledgeable about SAC and its commands. Users unfamiliar with SAC's capabilities probably will not be able to use SAC's command mode usefully to fix any problems. Therefore, expert users will probably find it most helpful.

5.9 OPERATING SYSTEM INTERACTION

SAC is not a stand-alone program but instead runs as an application program hosted by a UNIX-like operating system, such as a UNIX variant, a Linux system, or in the Cygwin environment under Windows. SAC is able to issue commands to the operating system that are the same as you would type them to the system itself. SAC provides SYSTEMCOMMAND (abbreviated SC) to do this. In its simplest form (more advanced uses will be discussed later), type SC followed by the operating system command:

```
SAC> sc date
Thu Oct 25 12:24:28 BST 2012
SAC> sc ls
dosect.make             picks-s2ks-grh          ppmap-make
fev2.stations           picks-s2ks-na           s2ks.picks-wrong
fev3.stations           picks-s2ks-new          s3ks.picks
fev4.stations           picks-s3ks-grh          s3ks.picks-orig
flt                     picks-s3ks-new          slant
SAC>
```

This runs the UNIX *date* command to provide the date and then the *ls* command to list the files in the current directory. The result in this case is typed on the screen.

A more powerful use of SYSTEMCOMMAND is to invoke a program that is part of a larger analysis scheme embedded in SAC. For example, a program could be written to do multitaper spectral analysis of a time series in a SAC file. The output of the program might be a SAC file containing the amplitude spectrum. After using SYSTEMCOMMAND to run the program, the resulting output file may be read into SAC and shown to the user. Packaging the analysis procedure inside the macro provides a new type of SAC command that uses external programs.

Some external programs need to have input provided to them other than through command line arguments. This is a limitation of the SYSTEMCOMMAND mechanism, which only provides for a single command line to be sent to the operating system. Of course, a SAC macro could write lines into an external file and then run the program with the file connected to standard input, but as a convenience, SAC provides a second mechanism to provide input to an external command or program. The $RUN command (only usable in a macro) names an external program to run and indicates that subsequent macro lines up to a $ENDRUN command are to be connected to the standard input of the program. For example, the *awk* program might be invoked to process a set of file names:

Ex-5.7

```
SAC> systemcommand cat tryawk
* Contents of macro "tryawk"
$run awk '{print @$1}'
&1,filename&
&2,filename&
&3,filename&
$endrun
SAC> datagen sub local cdv.[e,n,z]
SAC> m tryawk
cdv.e
cdv.n
cdv.z
SAC>
```

In this case, the macro was first listed using the *cat* command. Then SAC ran the macro, which provided the file names of the first three files in memory. The *awk* command printed the first blank delimited part of the file name (in this case, the whole name).

$RUN is useful because if the input to the program (the lines up to $ENDRUN) includes macro variables, blackboard variables, file variables, or expressions, they are evaluated before being passed to the program. This provides a simple way to get information internal to SAC passed to the program for its use. Note that because SAC uses $ to delimit macro variables, it was escaped (using @) to pass it unchanged to the *awk* command. In some cases, this might be required for command input lines as well.

5.10 LOOPING COMMANDS

SAC provides two looping constructs inside macros: DO and WHILE. (A loop is a sequence of commands that is repeated over and over by returning to the beginning after the end is

reached.) The difference between DO and WHILE is that DO repeats the loop a finite number of times, whereas WHILE repeats a loop for as many times as it takes for a condition associated with the command to become false. Both looping constructs bracket the repeated commands by ending them with ENDDO.

WHILE

The WHILE command is followed by a condition that is evaluated whenever the commands making up the loop (re)start. The next example uses a WHILE loop to prompt the user for a response until a valid response is given. This illustrates the potentially unlimited number of times the loop will be re-run, depending on the response of the user.

Ex-5.8

```
message "Fee fie foe fum" "I smell the blood of an Englishman"
message "What should I do now?"

* Ask for response and validate
setbb ok no                            ;* Sets value of bb var ok
while %ok% EQ no                       ;* Tests value of bb var ok
   setbb ans "_(reply 'respond kill or pause:')"  ;* Prompt, read
   if "%ans%" EQ _kill
      setbb ok yes
   elseif "%ans%" EQ _pause
      setbb ok yes
   else
      message "Invalid response"
   endif
enddo

* Act on response
if "%ans%" EQ _kill
   message "OK, quitting ..."          ;* Macro will terminate
   $KILL
endif
if "%ans%" EQ _pause
   message "Type @$RESUME to resume"   ;* Macro will be suspended
   $TERMINAL
   message "Resuming macro ..."        ;* Resumed at this point
endif
message "If he is alive or if he is dead"
message "I'll crush his bones to make my bread"
```

If the answer is “kill,” the macro stops running. If the answer is “pause,” the macro uses the suspend/resume facility to pause for later resumption. When resumed, the macro finishes.

WHILE READ

Another use of the WHILE command is for reading and parsing input from another program. WHILE can be used to read lines of input saved in a blackboard variable, presumably from output produced by an external program. To do this, use the READ <bb> verb following WHILE to indicate a blackboard variable to be read. A list of *macro variable* names follow that will be assigned blank-delimited words on each input line. The last variable will receive all of any words remaining, separated by blanks. (This is designed like the *sh* built-in *while read* command.) Note the distinction between blackboard variables (used as input) and macro variables (the output).

How to get input from another program will be introduced later (in Section 5.12 on advanced features of operating system interaction), but to see how looping is related to command input, suppose that one line of input was read and saved in the blackboard variable inp. The following WHILE loop will process it.

Ex-5.9

```
SAC> sc cat xample
* Contents of file "xample"
setbb inp "one two three and the rest"

while read inp a b c d   ;* bb variable read is %inp%
   message "a is $a$"    ;* note $a$-$d$ macro vars, not bb vars
   message "b is $b$"
   message "c is $c$"
   message "and d is '$d$'"
enddo

SAC> m xample
 a is one
 b is two
 c is three
 and d is 'and the rest'
SAC>
```

If there were more input lines in %inp%, the WHILE loop would repeat until they were exhausted.

Escaping from loops

No matter what the type of loop, SAC's BREAK command will jump out of the loop and resume processing using the command that follows ENDDO. When used inside a conditional statement, this provides a useful way to escape a loop before the full range of iteration possibilities are exhausted. Some uses of BREAK will be found in later examples.

DO

In contrast to WHILE, DO loops over a finite set of values. The values might be calculated from a range designated by a begin value, an end value and an increment (if different from one). Alternatively, DO can be given a list of explicit values to iterate over as blank-separated character strings. The strings might be literals or might be the result of a pattern match on files in the present directory.

The syntax for DO loop iterating over a numerical range is

```
DO <v> FROM <x> TO <y>
```

or

```
DO <v> FROM <x> TO <y> BY <z>
```

Here <v> is the name of a macro variable and <x>, <y> and <z> are numeric expressions (<z> is implicitly 1 if absent). The macro variable takes on the sequence of values starting with <x> and ending with <y>, running the commands up to ENDDO with each value assigned to <v>. The example below shows a macro that will calculate the first ten Fibonacci numbers using a DO loop.

Ex-5.10

```
SAC> sc cat dofrom
* Contents of macro file "dofrom"
setbb fib 1.0 prefib 0
do i from 0 to 10            ;* Loop macro variable value is i
   message "Fibonacci number $i$ is (BEFORE . %fib%)"
   setbb newfib (%fib% + %prefib%)     ;* Next Fibonacci num
   setbb prefib %fib% fib %newfib%     ;* Remember previous two
enddo

SAC> m dofrom
 Fibonacci number 0 is 1
 Fibonacci number 1 is 1
 Fibonacci number 2 is 2
 Fibonacci number 3 is 3
 Fibonacci number 4 is 5
 Fibonacci number 5 is 8
 Fibonacci number 6 is 13
 Fibonacci number 7 is 21
 Fibonacci number 8 is 34
 Fibonacci number 9 is 55
 Fibonacci number 10 is 89
SAC>
```

This macro illustrates a useful tactic to use when doing arithmetic with integers. SAC does all calculations as real numbers, even if the values are integers. Consequently,

(*before* . %fib%) is used to strip the trailing zeros from the floating point numbers before the result (the MESSAGE statement) is printed.

Another example of a loop that finds a power of 2 larger than a number given by an input prompt is as follows. This macro uses a loop to make increasingly large powers of 2 and then escapes from the loop when the smallest power bigger than the desired number is reached.

Ex-5.11

```
SAC> sc cat dobreak
* Contents of macro "dobreak"
setbb num "(reply 'Enter number:')"
do pow from 1 to 32      ;* Try powers of two
   setbb v (2 ** $pow$)
   if %v% GE %num%       ;* Test present one
      break              ;* Equals or exceeds number
   endif
enddo
message "Power of 2 larger than %num% is 2**$pow$ or %v%"

SAC>  m dobreak
Enter number:2096
 Power of 2 larger than 2096 is 2**12 or 4096.000
SAC>
```

The syntax for DO loop iterating over a selection of character strings is

```
DO <v> LIST <a> <b> <c> ...
```

or

```
DO <v> WILD DIR <d> <a> <b> <c> ...
```

Here <v> is a macro variable name, and <a> ... are character strings that <v> takes on successively before processing the commands up to the following ENDDO. In the case of a file pattern match, SAC scans the files in directory <d> for matches to the patterns <a> ... and collects the results into successive values for <v>.

The following example illustrates how a macro may be used to check whether the header of a SAC file is sufficiently populated to allow processing to proceed.

Ex-5.12

```
SAC> sc cat dolist
* Contents of macro "dolist"
do hdr list stla stlo evla evlo evdp scale ;* Hdr fields to check
   if UNDEFINED EQ "&1,$hdr$&"             ;* Check if set
      message "$hdr$ not set in file header"
   endif
enddo
```

```
SAC> funcgen seismogram                    ;* Generic data
SAC> m dolist
 scale not set in file header
SAC>
```

This example uses a built-in seismogram SAC has for testing, produced by the FUNCGEN command. The macro checks that the file header variables STLA, STLO, EVLA, EVLO, EVDP and SCALE are all set. If not, the macro produces an error message.

5.11 MACRO PARAMETERS

Until now, we have treated macros as files that perform a specific task when invoked by SAC. If the task requires user input, it comes from an expression that uses (*reply* ...). This section introduces more powerful capabilities of macros that make them resemble SAC commands in their own right.

The new capability is provided by arguments that may be appended to SAC macros. The arguments follow the macro name on the SAC command line. For a macro called example, the following commands to invoke the macro provide two arguments: a file name and a frequency.

In the first instance, the arguments are positional: the first argument is the file name and the second is the frequency.

```
SAC> m example KEVO.BHZ 2.5
```

The second example identifies the arguments by keyword that introduces a value:

```
SAC> m example file KEVO.BHZ freq 2.5
SAC> m example freq 2.5 file KEVO.BHZ  ;* order independent
```

The two examples are equivalent providing that the macro is expecting keyword parameters (note that keyword parameters are independent of position on the command line).

Arguments provided to a macro might be required or they might be optional. For example, a macro might require a file name to operate on, but it might be happy to assume that a frequency, if not otherwise given, takes on a default value.

Positional

Positional parameters are the simplest to use in a macro, so they will be introduced first. Inside the body of a macro, the macro variable 1 refers to the first argument that follows the macro name, 2 refers to the second, and so on. Thus the macro mppos, defined below, retrieves the values provided when the macro is invoked as follows:

Ex-5.13

```
SAC> sc cat mppos
* Contents of macro "mppos"
* Macro user provides two parameters:  SAC file and frequency
message "File name is $1$; frequency is $2$"
```

```
read $1$                                ;* Read SAC file
rmean; rtrend terse; taper w 0.05       ;* Prepare for filtering
lowpass corner $2$ npoles 2 passes 2    ;* Filter corner frequency

SAC> m mppos /tmp/ex-1 2                ;* Macro used here
 File name is /tmp/ex-1; frequency is 2
SAC>
```

Examine what happens if the second positional parameter (the frequency) is missing:

```
SAC> m mppos /tmp/ex-1                  ;* Second parameter missing
2?  5
 File name is /tmp/ex-1; frequency is 5

SAC>
```

The prompt "2?" requests that a value for the second parameter be provided before it is used. In this case, the number 5 is given and the macro proceeds. You can provide a default value for the second positional parameter of 5 if one is not provided by the user, as follows:

Ex-5.14

```
* Contents of macro "mpposm"
* Macro user provides two parameters:  SAC file and frequency
$default 2 5
message "File name is $1$; frequency is $2$"

read $1$                                ;* Read SAC file
rmean; rtrend terse; taper w 0.05       ;* Prepare for filtering
lowpass corner $2$ npoles 2 passes 2    ;* Filter corner frequency
```

The $DEFAULT command provides default values for macro parameters if a value is not provided when the macro is invoked. This says that the parameter called 2 (the second positional parameter) is to have the default value 5. Now see what happens when the macro is invoked in different ways:

```
SAC> m mpposm /tmp/ex-1                 ;* Second par omitted
 File name is /tmp/ex-1; frequency is 5

SAC> m mpposm                           ;* Both pars omitted
1?  /tmp/ex-1
 File name is /tmp/ex-1; frequency is 5

SAC>* Prompt for first parameter which lacks a default

SAC> m mpposm /tmp/ex-1 1
 File name is /tmp/ex-1; frequency is 1
SAC>* Explicit second parameter value overrides default
```

A default was only provided for the second positional parameter. If the first one is omitted it elicits a prompt when needed. Any value provided for the second parameter overrides the default.

Keyword

Most SAC commands do not expect arguments that depend on position. The BANDPASS command, for example, has keywords that introduce the low and high corner frequency, the number of poles and the number of passes. Similarly, SAC macros provide a way to define keywords that introduce arguments to the macro. The text following a keyword becomes the value of that parameter inside the macro. To introduce keywords to a SAC macro, use the $KEYS command at the beginning of the macro. The following redefines the previous macro using keyword rather than positional parameters:

Ex-5.15

```
SAC> sc cat mpkey
* Contents of macro "mpkey"
* Macro user provides file xxxx and freq yyyy
$keys file freq
message "File name is $file$; frequency is $freq$"

read $file$
rmean; rtrend terse; taper w 0.05
lowpass corner $freq$ npoles 2 passes 2

SAC> m mpkey file /tmp/ex-1 freq 5
 File name is /tmp/ex-1; frequency is 5
SAC> m mpkey freq 2 file /tmp/ex-2
 File name is /tmp/ex-2; frequency is 2
```

This example shows that keyword arguments can appear in any order, rather than being identified by position.

As with positional parameters, prompts are made for unassigned keyword parameters when their values are needed:

```
SAC> m mpkey file /tmp/ex-1
freq? 2
 File name is /tmp/ex-1; frequency is 2
```

In this case the prompt is clearer because it prints the keyword name. To provide a default value for a keyword, use the $DEFAULT command:

```
* Macro user provides file xxxx and freq yyyy
$keys file freq
$default freq 5                    ;* default frequency is 5
message "File name is $file$; frequency is $freq$"
```

```
read $file$
rmean; rtrend terse; taper w 0.05
lowpass corner $freq$ npoles 2 passes 2
```

Recursion

Macros can be recursive, which makes some algorithms simple to express. The macro listed below solves the Towers of Hanoi problem using positional parameters to describe a series of disk moves from one stack to another. Each stack is represented by a blackboard variable that contains a sequence of numbers representing a disk. The macro calls itself to move disks from one stack to another:

Ex-5.16

```
SAC> sc cat hanoi
* Towers of Hanoi:  3 stacks called A B C in BB vars A, B, C.
*  usage: m hanoi <n> <from-stack> <to-stack>
*    or   m hanoi setup <n> <stack>
if $1$ eq setup
   setbb A "" B "" C ""             ;* Clear out stacks
   do i from 1 to $2$
      setbb $3$ "%$3$% $i$"         ;* Add digit to stack end
   enddo
   message "Initial arrangement:  %A% | %B% | %C%"
elseif $1$ eq 1
   setbb tmp "(item 1 %$2$%)"       ;* Copy from stack top item
   setbb $2$ "(items 2 END %$2$%)"  ;* Remove from stack top item
   setbb $3$ "%tmp% %$3$%"          ;* Add item to top of to stack
   message "Move %tmp% from $2$ to $3$:  %A% | %B% | %C%"
else
   m hanoi (int ($1$ - 1)) $2$ (delete $2$ (delete $3$ 'ABC'))
   m hanoi 1 $2$ $3$
   m hanoi (int ($1$ - 1)) (delete $2$ (delete $3$ 'ABC')) $3$
endif
SAC> m hanoi setup 3 A
 Initial arrangement:   1 2 3 |  |
SAC> m hanoi 3 A C
 Move 1 from A to C:  2 3 |  | 1
 Move 2 from A to B:  3 | 2 | 1
 Move 1 from C to B:  3 | 1 2 |
 Move 3 from A to C:    | 1 2 | 3
 Move 1 from B to A:  1 | 2 | 3
 Move 2 from B to C:  1 |  | 2 3
 Move 1 from A to C:    |  | 1 2 3
```

5.12 ADVANCED OPERATING SYSTEM INTERACTION

We met SYSTEMCOMMAND earlier as a way to send commands to the operating system running SAC. The command is run by the operating system while SAC waits for the result. When completed, a user or a SAC macro could then retrieve the result.

SYSTEMCOMMAND provides a further feature that, when combined with WHILE READ, provides a powerful and flexible way to retrieve output from a program and process it inside SAC. Output from any command can be retrieved and put into a blackboard variable, which is then available for processing inside SAC. The syntax of SYSTEMCOMMAND when used in this way is

```
SYSTEMCOMMAND TO <v> cmd
```

Here <v> is the name of a blackboard variable whose value will become the result of the output from cmd. This macro will parse the result of the *date* command to write out the time:

Ex-5.17

```
SAC> sc cat demosc
* Contents of the macro "demosc"
sc to out date                          ;* date output to bb var out
message "Time now is: (ITEM 4 %out%)" ;* select 4th field in %out%

SAC>  m demosc
 Time now is: 09:51:29
SAC>
```

Additionally, SAC preserves line breaks between output lines read from the command. This means that using the WHILE READ command in a loop will process the lines produced as output. The following macro shows how SAC can parse the result of the *ls* command to list the names and sizes of files:

Ex-5.18

```
SAC> sc ls -l /tmp/do*            ;* output from ls command
-rw-------  1 geo-g4  wheel  210 Jun 17 09:18 /tmp/dobreak
-rw-------  1 geo-g4  wheel  179 Jun 17 08:55 /tmp/dofrom
-rw-------  1 geo-g4  wheel  309 Jun 17 10:06 /tmp/doscls

SAC> sc cat doscls
* Contents of macro "doscls"
sc to lsout ls -l /tmp/do*        ;* output saved in bb var lsout

while read lsout fperm flink fowner fgroup fsize fmm fdd ftt fnm
   * Field 1 of output [permissions] assigned to fperm
   * Field 2 of output [links] assigned to flink
   * Field 3 of output [user name] assigned to fuser, etc.
   message "File $fnm$ size $fsize$"
enddo
```

```
SAC>  m /tmp/doscls
 File /tmp/dobreak size 210
 File /tmp/dofrom size 179
 File /tmp/doscls size 309
```

Another handy use for retrieving system command output is to generate temporary file names that will not conflict with names used by other computer users. This takes advantage of the shell's built-in variable $$, which expands into the process ID of the shell itself. A unique temporary file name can be obtained in the following way:

```
sc to scr echo /tmp/temp@$@$.sh
```

Note the use of the escape character @ to prevent interpreting $ as a macro variable indicator. After this, references to the blackboard variable `%scr%` will provide the name of a temporary file that will not conflict with other users.

The final example is a macro that will check whether file names with particular prefixes and suffixes exist. The macro defines a shell script designed to test whether a particular file exists. It then builds a list of file names as input for that shell script. The output is a list of file names and existence verdicts: "yes" if it exists, "no" if not. The macro reads this output and the list of files and prints out whether each exists.

Note that there are many times when characters especially significant to the shell must be escaped to prevent their interpretation by SAC.

Ex-5.19

```
SAC> sc cat doexist
* Contents of macro "doexist"
$keys pfx sfx
$default sfx BHE BHN BHZ                ;* Default suffix list
sc to scr echo /tmp/temp@$@$.sh         ;* Temp script name scr

* Copy text to create a temp. shell script to check if file exists
*   Note need to use escape characters for shell script syntax.
$run cat - > %scr%
# This shell script reads a file name and checks whether it exists
#   It echos the file name and adds "yes" if it exists or "no"
while read fn ; do
   test -f \@$fn @&@& echo \@$fn yes || echo \@$fn no
done
$endrun

sc to finp echo /tmp/temp@$@$.in        ;* Temp input file finp
do s list $sfx$                         ;* Iterate on suffix list
   sc echo $pfx$*.$s$ >> %finp%         ;* Add name to input
enddo

sc to inp sh %scr% < %finp%             ;* Run script on input

while read inp fn exist                 ;* Process script output
```

```
   message "File $fn$ exists? $exist$"   ;* Report file existence
enddo

sc rm %scr% %finp%                       ;* Remove temporary files

SAC> m doexist pfx /data/hinet/2005/vel/EDSH sfx LE LN LZ LR LT
 File /data/hinet/2005/vel/EDSH.LE exists? yes
 File /data/hinet/2005/vel/EDSH.LN exists? yes
 File /data/hinet/2005/vel/EDSH*.LZ exists? no
 File /data/hinet/2005/vel/EDSH.LR exists? yes
 File /data/hinet/2005/vel/EDSH.LT exists? yes
SAC>
```

CHAPTER

SIX

Accessing SAC functionality and data from external programs

Despite the broad range of utility that SAC provides, at times it may be necessary to use external applications to augment its capabilities. In addition, it is sometimes desirable to access the functionality of SAC without interacting manually with the program, for example to include it in a longer processing workflow. In this chapter we will describe techniques and give examples of how to achieve both these ends.

Note that details of the languages and applications for which examples are shown are beyond the scope of this book. The reader should seek more specialized books for that material.

6.1 AUTOMATING SAC EXECUTION

Running SAC from the shell

Executing SAC

While a decent amount of batch processing is possible within SAC using its built-in macro language (see Chapter 5), it is often useful (for example, to access functionality built into the operating system) to run SAC using scripting languages provided by the shell (under UNIX-like environments, examples include *bash* and *tcsh*).

One way is by using startup files (see Section 4.10). After the commands in it are executed, SAC then enters interactive mode. Because (usually) there is no need for an interactive phase when scripting, it is helpful to make the last line a `QUIT` command to terminate SAC and allow control to return to the shell.

So, if the file `MyCommandsFile` contains the commands

```
r MyStation.BHN MyStation.BHE ; * read in horizontal components
rotate to gcarc               ; * rotate them to radial-transverse
w MyStation.BHR MyStation.BHT ; * output as new files
quit                          ; * terminate SAC
```

on executing

```
sac MyCommandsFile
```

SAC will read in the specified data files, perform the rotation, write out the new traces and terminate.

Errors and SAC output

If an error is encountered during the execution of the SAC macro, SAC will halt to the interactive prompt. Output will be, as in a normal SAC session, directed to the standard output (the normal UNIX output stream) by default. This can then be captured or disposed of using standard UNIX command line syntax:

```
sac MyCommandsFile > /dev/null ; # execute sac and dispose
                                   of the terminal output.
```

Graphics

Unless SAC interactive commands are needed (for example, for time picking), it is usually sensible to direct any required graphical output to SAC's file-based plotting devices (such as the `sgf` device, see Section 4.4) to prevent SAC from locking up the display as it generates plots. For example, a file containing the commands

```
r MyStation.BHN MyStation.BHE ; * read in horizontal components
rotate to gcarc               ; * rotate them to radial-transverse
begindevices sgf              ; * open the SGF graphics device
ylim all; plot1               ; * plot radial vs transverse
enddevices sgf                ; * terminate the SGF graphics device
quit                          ; * terminate SAC
```

will create the file `f001.sgf`, which is a SAC graphics format file containing a plot of the radial and transverse components.

Automation of SAC execution in the shell using scripting

The utility of being able to control SAC from the shell command line is the ability to execute it repeatedly in automated shell scripts. This enables blending shell capability (for example, file handling) with SAC's functionality. It should be noted that this is not an efficient method of processing large amounts of data because shell scripts (being interpreted languages) are slow, and there is additional overhead associated with repeatedly starting and terminating SAC. However, for many applications where smallish amounts of data are involved, or the task only needs performing once, these drawbacks are heavily outweighed by its convenience.

Example: Batch processing of data files

A simple example of this is to use shell scripts to enable batch processing of large numbers of data files. While this is also possible to do using SAC macros (see Chapter 5), it is often more convenient to use the capability provided by shell scripting.

Suppose we have a directory structure containing event selected data for a network of stations. The root directory UB contains a directory for each station (UB01 to UB05), each of which contains a folder for each event recorded (EV01 to EV10). These event folders contain three SAC files containing the data from each component for that event:

```
$ ls -Rx UB/*/*/*.SAC
UB/UB01/EV01/E.SAC  UB/UB01/EV01/N.SAC  UB/UB01/EV01/Z.SAC
UB/UB01/EV02/E.SAC  UB/UB01/EV02/N.SAC  UB/UB01/EV02/Z.SAC
...
UB/UB05/EV09/E.SAC  UB/UB05/EV09/N.SAC  UB/UB05/EV09/Z.SAC
UB/UB05/EV10/E.SAC  UB/UB05/EV10/N.SAC  UB/UB05/EV10/Z.SAC
```

To generate radial and transverse components for each of these we create a simple shell script that will traverse the directory structure and run a simple set of SAC commands in each directory:

Ex-6.1

```
#!/bin/bash
# create a SAC macro to rotate the data,
# and save in /tmp
cat << END > /tmp/RotateMacro
   r N.SAC E.SAC   ; * load the data
   rotate to gcarc ; * rotate to R-T
   w R.SAC T.SAC   ; * write out
   quit
END
# outer (station) loop
for station in UB/UB??
do
   cd $station
   # inner (event) loop
   for event in EV??
   do
      cd $event
      echo "Rotating " $event " for " $station
      # run SAC
      sac /tmp/RotateMacro > /dev/null
      cd ..
   done
   cd ../..
done
# End of script.
```

Note that in this example the SAC macro is generated at the start of the script. Not only is this neater (keeping everything in a single file), it allows use of the tools provided by the shell script to manipulate the SAC macro. An alternative script that achieves the same result as that given above looks like this:

Ex-6.2

```
#!/bin/bash
# outer (station) loop
for station in UB/UB??
do
   # inner (event) loop
   for event in $station/EV??
   do
      # create SAC macro
      echo $event
      echo "r "$event"/N.SAC "$event"/E.SAC" > /tmp/RotateMacro
      echo "rotate to gcarc"  >> /tmp/RotateMacro
      echo "w "$event"/R.SAC "$event"/T.SAC" >> /tmp/RotateMacro
      echo "quit"  >> /tmp/RotateMacro
      # run SAC
      sac /tmp/RotateMacro > /dev/null
   done
done
# End of script.
```

In this version, rather than traversing the directory structure, the script generates a new macro for each set of files and executes SAC from the root directory. This ability to create and modify macros during execution of the script significantly increases the range and capability of automation. Although beyond the scope of this book, it is possible to use shell scripting to enable parallel processing of SAC data by spawning multiple processes within a shell script.

Example: Extracting information from SAC files

This technique can also be used to extract information from a set of SAC files for external processing. Suppose, using the dataset described above, we have picked the direct S-wave arrival in each of the transverse component seismograms and stored it in the `T1` header field (see Section 4.5). We now want to extract the time pick and the epicentral distance from the event to the station from the files to a single file for plotting. This is achieved by scripting:

Ex-6.3

```
#!/bin/bash
# create a SAC macro to rotate the data,
# and save in /tmp
cat << END > /tmp/ExtractPick
   r T.SAC
   message '&1,gcarc& &1,t1&'
```

```
   quit
END
# create an empty pick file
cat /dev/null > pickfile
# outer (station) loop
for station in UB/UB??
do
   cd $station
   # inner (event) loop
   for event in EV??
   do
      cd $event
      # run SAC, and put the last line of output
      # into the pick file
      sac /tmp/ExtractPick | tail -1 >> ../../../pickfile
      cd ..
   done
   cd ../..
done
# End of script.
```

This results in a two-column text file (called `pickfile`) containing the epicentral distance and corresponding time. This could then be, for example, plotted using any of a variety of software.

6.2 ACCESSING SAC DATA IN EXTERNAL PROGRAMS

The relative simplicity of the SAC data format has made it popular for storing seismic data, especially in a working, as opposed to archival, format. This makes it simple to include, for example, independent applications into a SAC processing workflow. (Note, in the following we only discuss SAC binary files.)

Accessing SAC data from Fortran using the `sacio` library

The simplest way to interact SAC data with user applications in Fortran is to use the `sacio` library. The development of this and other libraries is integrated with the development of SAC itself, and thus any future changes in SAC format will be transparent to the user (since the interfaces will not change), making code maintenance easier.

The `sacio` library provides a level of abstraction from the SAC file. It provides the following callable subroutines.

- `RSACH`: This subroutine instructs the library to load the header (only) of a specified SAC format data. The header information is stored within the library.
- `RSAC1`: This subroutine instructs the library to load a specified (evenly spaced) SAC format data file. The trace data (length and values), the initial time and the sampling rate are returned. The other header information is retained within the library.
- `RSAC2`: This is the equivalent of `RSAC1` for unevenly spaced files.

- GETFHV: Once SAC header information has been loaded using RSACH, the header values can be queried by name from the library (see HELP HEADERS for the available variables). GETFHV fetches the floating point headers (for example, time headers: A, F and T0–T9; and event location information: EVLO, EVLA and EVDP).
- GETNHV, GETKHV and GETLHV: As above, but for querying integer, character and logical header values respectively.
- SETFHV, SETNHV, SETKHV and SETLHV: These are counterparts of the GETxHV family, which store user-specified values of the appropriate type to the header retained in the library.
- WSAC0: Output a SAC file, combining the header values currently held in the library and one or two user-supplied arrays (for evenly and unevenly spaced formats, respectively).
- WSAC1: This is a simpler write function to write an evenly spaced file with a minimum header which does not require a previously loaded trace. This is useful for quickly outputting temporary or quality-control files or files for which the header no longer applies.
- WSAC2: This is the equivalent of WSAC1 for unevenly spaced files.

The complete calling sequence for these routines is available in the SAC help system. Note that the model adopted by the sacio library means that only one SAC file at a time can be interacted with. A second RSAC1 call will overwrite the header information currently kept by the library. For this reason, concurrent interaction with multiple traces requires the user to extract (through GETxHV calls) the header values they need and store them in variables in their own code. Furthermore, for efficiency reasons, sacio library routines do not perform extensive error correction or consistency checking of the files during read/write operations.

Example: Doubling of trace amplitudes

The following Fortran code demonstrates the use of these routines to double the amplitudes in a seismogram stored in a SAC data file:

Ex-6.4

```
program thedoubler
   implicit none
   integer, parameter :: nmax = 10000
   real :: ampl(nmax),beg,dt
   integer :: npts,nerr,ip

   ! Read in the data file
   call rsac1('data.sac',ampl,npts,beg,dt,nmax,nerr)

   ! loop over the data points, doubling each
   do ip=1,npts
      ampl(ip) = ampl(ip) * 2.0
   enddo

   ! set a user character header
   call setkhv('KUSER0','DOUBLED!',nerr)
```

```
   ! write the altered file back to
   !    disk under a different name.
   call wsac0('doubled.sac',ampl,ampl,nerr)
end program thedoubler
```

It is compiled and linked with the `sacio` library with a command like:

```
gfortran -o thedoubler thedoubler.f90 /usr/local/lib/sacio.a
```

(The exact form of this command will depend on the details of the installation of SAC, the compiler used and so on.) When executed, it reads in `data.sac`, loops over the array, doubling each entry and setting a character header variable (`KUSER0`) to contain the text "DOUBLED!", and then writes the new file back to disk with the name `doubled.sac`.

`sacio90`: object-oriented SAC data interaction in Fortran

Modern versions of Fortran (Fortran90 and later) provide facilities for a more object-oriented approach to programming than was possible in earlier incarnations. This has a number of advantages for the convenience of handling SAC files. In the standard `sacio` library, seismic data are separated from associated metadata in their headers. The library `sacio90` provides data structures and routines to provide access to SAC data in a more object-oriented way. Data and associated headers are stored in a single structure, making handling complete seismograms much more convenient. It also makes concurrent handling of multiple traces simpler.

`sacio90` uses the `sacio` behind the scenes to maintain compatibility. Extensive documentation of this library is beyond the scope of this book, but an example of a program using this functionality might look like this:

Ex-6.5

```
program thedoubler
   use sacio90 ! import the SACIO90 module
   implicit none

   type (SAC1) :: trace ! SAC time-series file structure

   ! Read in the data file
   call sacio90_read('data.sac',trace)

   ! double all data points
   trace % y(1:trace % npts) = &
      trace % y(1:trace % npts) * 2.0

   ! set a user character header
   trace % kuser0 = 'DOUBLED!'
```

```
   ! write the altered file back to disk under
   ! a different name.
   call sacio90_write('doubled.sac',trace)
end program thedoubler
```

This program replicates the functionality of the previous example. The seismogram, including all of its associated headers, is returned to the user in the structure `trace`. The header variables `KUSER0` and `NPTS` and the data are stored in the fields of this structure.

Other languages

Many libraries offering access to and manipulation of SAC data files have been written over the years by the seismological community in a number of different languages. Since the SAC data format is relatively simple (compared, for example, to SEED), it is relatively easy to code one's own routines to provide simple read/write access to SAC data files. Several libraries are included in the code library accompanying this book (for example, for accessing SAC files in Python and MATLAB), but many more are distributed for other languages. This allows access to functionality provided by these languages, such as more advanced graphics capability.

Example: Variable density plot in MATLAB

The following code demonstrates using the 3D plotting facilities of MATLAB to generate a variable density plot of a set of seismograms recording a seismic phase arriving at an array. The resulting plot is shown in Figure 6.1. This uses the library MSAC.

Ex-6.6

```
% read a set of traces into an array of SAC structures.
RS = msac_mread('*.BHE') ;

% build normalised amplitude, time and distance matrices.
for itr = 1:length(RS)
   A(:,itr) = RS(itr).x1./(RS(itr).depmax-RS(itr).depmin) ;
   D(:,itr) = ones(1,RS(itr).npts).*RS(itr).gcarc ;
   T(:,itr) = (0:RS(itr).npts-1).*RS(itr).delta + RS(itr).b ;
end

% sort matrices into order of increasing epicentral distance.
[~,ind] = sort([RS(:).gcarc]) ;
AS = A(:,ind) ; TS = T(:,ind) ; DS = D(:,ind) ;

% plot as greyscale surface plot.
surf(DS,TS,AS,'LineStyle','none')
colormap(gray) ; caxis([-0.75 0.75]); axis tight
xlabel('Epicentral distance (deg)') ; ylabel('Time (s)')
shading interp ; view(0,90) ;
```

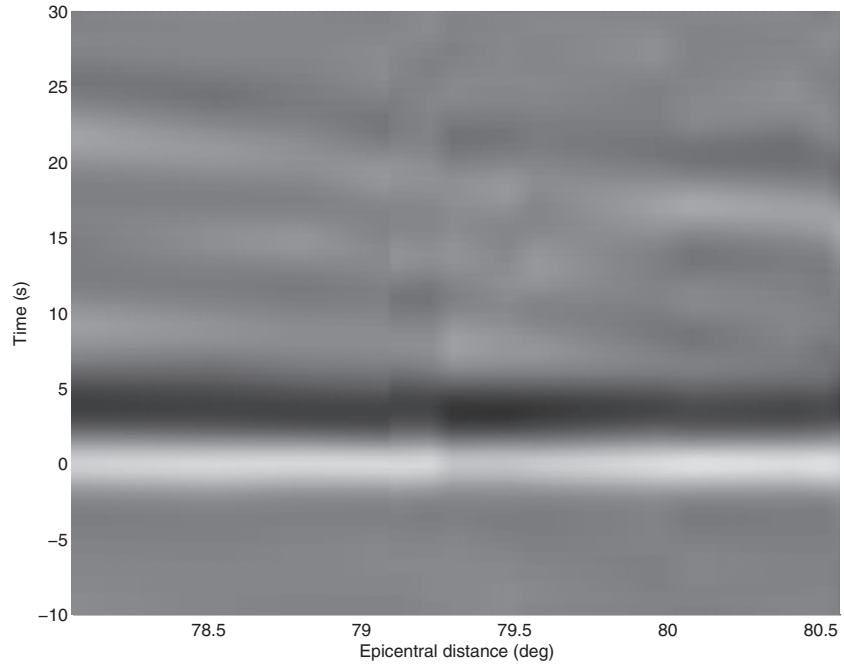


Figure 6.1 Various community-supported libraries exist for interacting with SAC in other programming languages. This example shows a variable density plot of a set of seismograms recording a teleseismic S-phase arriving across a regional array. This is generated in MATLAB, using the library MSAC to interact with SAC-formatted seismic data (see text for source code).

6.3 ACCESSING SAC FUNCTIONALITY IN FORTRAN PROGRAMS

SAC is often used as a preprocessor for seismic data destined for some other external analysis. This is often done because SAC represents a trusted (or, at least, very well tested) method for doing simple processing (filtering, for example). While access to this SAC functionality is possible using file input/output and shell scripting (see Section 6.1), for many applications the performance cost of this is prohibitive. However, SAC provides a library interface to its filtering capabilities (see Section 4.9), which are among its most used functionality. As with the `sacio` library, this is callable from the user's own Fortran programs and is implemented in the externally callable subroutine `xapiir` (see `HELP XAPIIR`).

Example: Bandpass filtering a seismogram

The following Fortran code implements a simple Butterworth bandpass filter on a seismogram:

Ex-6.7

```
! Demonstrate a simple bandpass Butterworth filter using
! the XAPIIR library distributed with SAC.
program simple_filter
```

```
   implicit none
   integer, parameter :: nmax = 10000
   integer :: nlen, npts, nerr
   real*4 :: data(nmax)
   real*4 :: beg, dt, unused

   ! filter parameters
   integer :: n_order, n_pass
   real*4 :: flo, fhi

   ! Read in the data file
   call rsac1('raw.sac',data,npts,beg,dt,nmax,nerr)

   ! Set up filter parameters
   n_order = 2   ! two pole
   n_pass  = 2   ! two pass
   flo     = 1.0 ! low cut-off
   fhi     = 5.0 ! high cut-off
   unused  = 0.0 ! does not apply for Butterworth.

   ! call the filter library, specifying a Butterworth
   ! bandpass filter.
   call xapiir(data,npts,'BUTTER', &
      unused,unused,n_order,'BP',flo,fhi,dt,n_pass)

   ! write the filtered back to disk.
   call wsac0('filt.sac',data,data,nerr)

end program simple_filter
```

This is equivalent to the following commands in SAC:

```
SAC> r raw.sac
SAC> bandpass butter corners 1.0 5.0 npoles 2 passes 2
SAC> w filt.sac
```

There is an ongoing effort to include more SAC functionality in the externally callable libraries.

CHAPTER
SEVEN

Graphical data annotation

7.1 PLOT ANNOTATION

Seismograms

For SAC, the basic display element is a seismogram plotted with time along the horizontal axis and the sampled quantity along the vertical axis. SAC's basic plotting commands, PLOT, PLOT1 and PLOT2, supply axis tick marks numbered to show the relevant scales and connect data points with lines. SAC draws tick marks around all four sides of the plot to facilitate quantitative use of the data in the plot.

Tick marks, axes and borders

SAC uses a good default algorithm that chooses tick mark intervals and numbering based on the range of values to be plotted. If the result is unsatisfactory, tick mark intervals may be explicitly set using the XDIV and YDIV commands for the horizontal and vertical axes, respectively. The number or spacing of the tick marks may be specified using the keywords INCREMENT or NUMBER. To reset the default settings, use the NICE option.

The TICKS command selectively turns off the tick marks on each of plot's four sides using the keywords TOP, BOTTOM, LEFT and RIGHT, respectively. The omission of tick marks results in clearer plots, but they lose their quantitative clarity. TICKS is mostly used to make publication-quality plots for inclusion in a talk or a publication.

Grid lines across the plot (like graph paper) may be useful in some circumstances but are distracting due to the resulting clutter. To draw grid lines on plots, use the GRID, XGRID and YGRID commands; they are off by default. A solid or dotted grid line may be chosen in either or both of the X and Y directions. Each command recognizes the keywords ON or OFF to control their use and SOLID and DOTTED to express their form. The keywords may be

used with any of the three commands. GRID controls both the X and Y axis grids, whereas XGRID and YGRID affect grid drawing on one coordinate only.

The default plot that SAC draws is of a trace surrounded by a box. The tick marks or grid lines extend from the border into the box. The X and Y coordinate axes form part of this box and may be selectively omitted by the AXES command. For example,

Ex-7.1

```
SAC> FUNCGEN SEISMOGRAM
SAC> BW 1; PLOT1
SAC> AXES ON ONLY LEFT BOTTOM; TICKS OFF ALL
SAC> BW 2; PLOT1
```

will display a seismogram with axes labeled only on the left and bottom. Figure 7.1 shows an example of the resulting plots with and without tick marks.

If all axes and tick marks are omitted, SAC produces a bare plot consisting of the data trace and any identifying labels. If a box is still desired around the plot, the BORDER command may be used to force one to be drawn. By default, there is no border because it is formed from the axes and tick choices.

Labels may be provided for the X and Y axes using the XLABEL and YLABEL commands and for the entire plot using the TITLE command. A label can be displayed or suppressed with the ON or OFF keywords and set by giving a string of text enclosed in single or double quotes. The size of the text may be given by specifying a SIZE of either TINY, SMALL, MEDIUM or LARGE. The position of the label is specified by giving a LOCATION of TOP, BOTTOM, LEFT or RIGHT. To restore label characteristics to their value prior to being changed, use the PREVIOUS option. With three labels to choose from, fairly detailed labeling is possible (and potentially confusing).

Notes can also be placed inside the data area of the plot itself, using the PLABEL command. Up to five notes can be added to a plot this way. The notes are numbered 1–5 and may be individually turned ON or OFF or set by enclosing text in single or double quotes. The text size may be specified as for the other labeling commands using SIZE. The text position is set relative to the previous note by using the keyword BELOW, or by giving a POSITION <x> <y>, where <x> and <y> are given in fractions of the window size (between 0 and 1). The note's text may be placed at a non-horizontal angle that follows <y>; the angle is in degrees clockwise from horizontal.

Publication-quality plots often require borders to be omitted. Turning off axes and tick marks will usually achieve this. It may occasionally be useful for a box to be drawn around the data in the plot nonetheless. The BORDER command may be used for this. Normally off, an explicit border may be plotted, suppressed, or restored to its previous state by using the keywords ON, OFF and PREVIOUS.

Trace information

Any seismogram generally looks like any other. Consequently, information on a seismogram plot is crucial to identify the data under scrutiny. By default, SAC adds information to each trace based on what is recorded in the file header (see Fig. 7.1). The items included (if known) are the event name, the station component names, and the zero date and time of the trace. If none of these are known, SAC uses the file name.

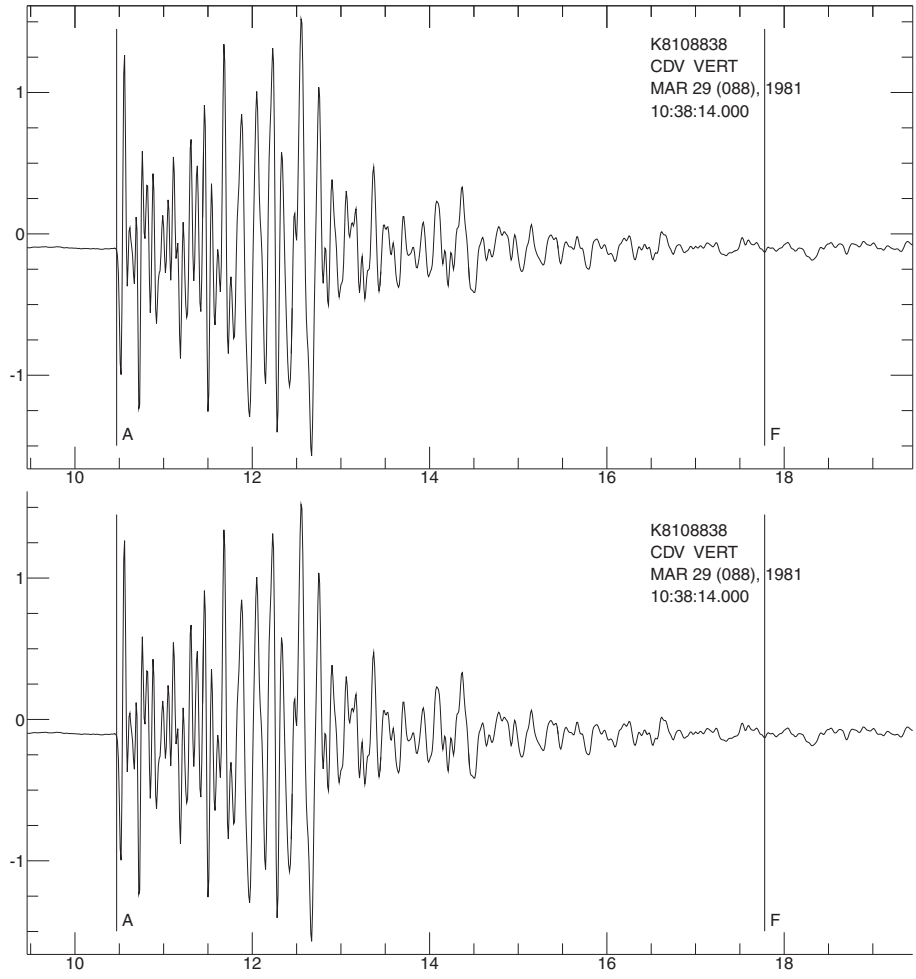


Figure 7.1 A basic data plot by PLOT1. (*top*) The plot contains axes on the left and bottom (numerically labeled) and tick marks on the top and right. (*bottom*) Turning off tick marks eliminates the box around the data. Both panels contain a file ID at the upper right consisting of an event name, a station and component name, and a zero time for the trace. The vertical lines indicate time picks stored in the file header. The labels next to the pick lines indicate the default pick name.

The information, its position and its display format can be changed using the FILEID command. The options ON or OFF turns FILEID on or off. The information TYPE can be either DEFAULT, NAME for the file name, or TYPE <list>, where <list> refers to a set of file header variable names to be included. The location in the trace window is given by a LOCATION of UR, UL, LR, LL, where U and L indicate the upper or lower side of the window, and R and L indicate its left or right side. The keyword FORMAT giving the form of information display is either EQUALS (a name = value format), COLON (a name: value format), or NONAMES (simply the value).

Picks

Picks associated with the trace that are recorded in the header (A, F, O, T0, ..., T9) may also be labeled. Within a seismogram, a pick is typically shown as a vertical line at the pick time (see Fig. 7.1), less commonly as a cross, or rarely as a horizontal line at the level of the data point nearest to the time pick. A label may be associated with a pick (for example, "S-wave" might be associated with the pick T0) and, when the pick is displayed, that label is shown alongside of the pick. If no label is given for a pick, its name is shown instead. Use the CHNHDR command to set the pick labels in the file header (KA, KF, KO, KT0, ...).

Pick display is under control of the PICKS command. The ON and OFF options control whether they are shown on a trace. You can set the height and width of the picks using the WIDTH <v> and HEIGHT <v> options. Here, <v> is a numeric value between 0 and 1 representing the fraction of the plot size. By default, the type of all picks is a vertical line, but this can be changed by naming the pick and following it with a type of VERTICAL, CROSS or HORIZONTAL.

Composite plots

SAC's basic display is a time series plot. Depending on the plotting command used, there might be one or more traces in a plot, but they generally share the same time coordinate axis. Often it is useful to show two traces in the same plot with different time axes, or even with different independent variables. For example, one might simultaneously wish to see a seismogram and its spectrum in the same plot.

SAC provides a facility to combine many different types of plots in a single display. Two new concepts underlie this facility. The first is the idea of a *frame*, which collects a group of plots into a single graphical display. When a frame opens, the display is cleared, and when a frame closes, the plot group is shown on a display. Inside an open frame, a plot is made in a *viewport*. One positions the viewport in a location in the frame and makes a plot using one of SAC's plotting commands. Changing the viewport's position and drawing another plot fills the frame with the desired information. Closing the frame realizes the individual plots on the display.

The commands BEGINFRAME and ENDFRAME open and close frames. Once SAC sees a BEGINFRAME command, no plots will be shown until an ENDFRAME appears. Thus these commands must be used in pairs to delimit a display.

A viewport is defined by giving its extent in the X (horizontal) and Y (vertical) directions. The extents are in fractions of the window size and range from 0 to 1. This convention means that, if a window is shrunk or expanded, the viewport correspondingly shrinks or expands. The X and Y viewport extents are set using the commands XVPORT and YVPORT, respectively.

Example

The following is an example of a composite plot that illustrates the ideas of a frame and a viewport and shows how labels may be used. Figure 7.2 shows the resulting plot.

Ex-7.2

```
SAC> FUNCGEN SEISMOGRAM  ;* create test data
SAC> CUTIM A -0.2 N 512  ;* window data in memory
```

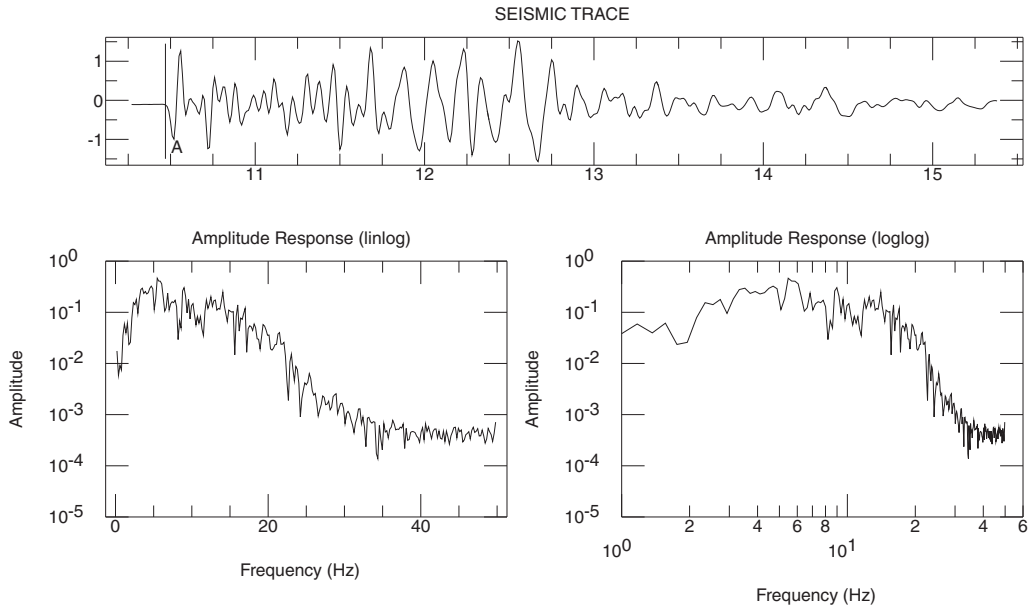


Figure 7.2 A multi-plot display example. The plot consists of three panels, each described with a viewport. Each panel has axis labels associated with it and is displayed inside of its corresponding viewport.

```
SAC> BEGINFRAME          ;* turn off automatic framing
SAC> XVPORT .1 .9        ;* define viewport and options
SAC> YVPORT .7 .9
SAC> TITLE 'SEISMIC TRACE'
SAC> FILEID OFF          ;* turn off fileid and qdp option
SAC> QDP OFF
SAC> PLOT                ;* plot the trace

SAC> XVPORT PREVIOUS     ;* restore viewport
SAC> YVPORT PREVIOUS
SAC> TITLE PREVIOUS      ;* restore title

SAC> FFT WMEAN           ;* take transform of data
SAC> XVPORT .1 .45       ;* second viewport and options
SAC> YVPORT .15 .55
SAC> TITLE 'Amplitude Response @(linlog@)'
                         ;* use of escape character @ prevents
                         ;* parsing of linlog as a function name
SAC> LINE FILL OFF       ;* suppress any fill of spectral plots
SAC> YLIM 1E-5 1
SAC> PLOTSP AM LINLOG    ;* plot the amplitude

SAC> XVPORT PREVIOUS     ;* restore viewport
SAC> TITLE PREVIOUS      ;* restore title
```

```
SAC> XVPORT .55 .9       ;* third viewport and options
SAC> TITLE 'Amplitude Response @(loglog@)'
SAC> XLIM 1 60
SAC> PLOTSP AM LOGLOG    ;* plot amplitude again
SAC> XVPORT PREVIOUS     ;* restore viewport
SAC> YVPORT PREVIOUS
SAC> TITLE PREVIOUS      ;* restore title
SAC> ENDFRAME            ;* resume automatic framing

SAC> LINE FILL PREVIOUS  ;* reset changes to previous values
SAC> FILEID PREVIOUS
SAC> XLIM PREVIOUS
SAC> YLIM PREVIOUS
```

7.2 ANNOTATING PLOTS WITH GRAPHICAL ELEMENTS

Changing elements of SAC's standard plots and grouping plots together to form a larger display are two ways outlined so far to form specialized displays. SAC provides a more powerful suite of commands for building up elaborate displays that do not necessarily use SAC's data plotting facilities.

Figure 7.3 shows an example of a display produced by SAC. The display, showing a clock, is built up from a series of plotting elements assembled in a particular way. The elements comprising the clock are circles (the face outline and the hand boss), arrows (hands), lines (hour markers) and text (numerals and manufacturer). SAC's plotting facilities include these (and more) elements that may be assembled into more elaborate displays or may be used to annotate any of SAC's basic plots within the same frame. The SAC command that draws these elements is PLOTC. Because the assembly of the plot elements can be quite complex, the description is contained in auxiliary files that are read by PLOTC to create the plot. PLOTC

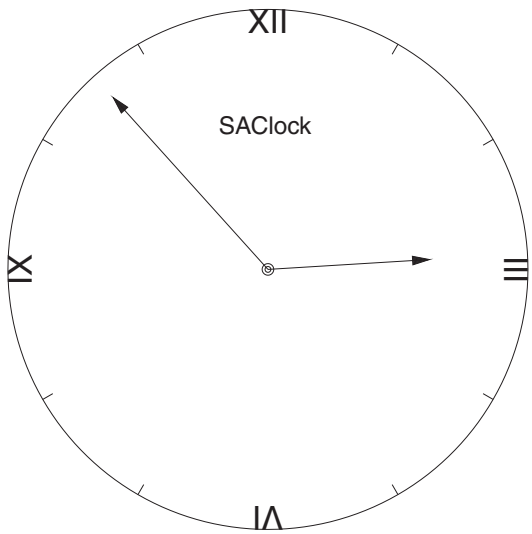


Figure 7.3 An example of a SAC display that is not time series data. The display consists of graphical elements (circles, lines, arrows, text) assembled in a way to form a clock that tells the time.

can be used to create auxiliary files for display annotation, usually at a preliminary stage in the design phase of a complex display when a rough placement of plot elements is made. Subsequently, the file is edited to refine the display before use in routine seismic analysis.

The files used by PLOTC to make displays must have names ending with either .PCF or .PCM. Files with the .PCM suffix are confusingly called macro files, but they are distinct from SAC command macros in that they only contain display descriptions. PLOTC makes the distinction because macros contain groups of graphical elements that may be scaled and rotated as a group and repetitively placed in a display. They only get invoked in the course of reading from a .PCF file while PLOTC makes a display. The suffixes are implied when these files are referred to. For example, when making a display described by the file MYCLOCK.PCF, invoke the PLOTC command via

Ex-7.3

```
SAC> PLOTC REPLAY FILE MYCLOCK
```

Graphical elements

SAC's PLOTC command provides the following graphical elements:

- arrow;
- set of tick marks;
- circle;
- line;
- polygon (2–9 sides);
- sector of a circle;
- rectangle;
- line of text;
- block of text;
- group of other elements (called a macro).

Additional items that appear in display descriptions are comments (which are ignored), parameter assignments (to set line styles and polygon sides, for example), element placement information and the end-of-plot indicator.

SAC's plot windows typically do not have a 1:1 aspect ratio, which distorts the shape of the graphical element. If shape is important, use the VSPACE command to set the view space aspect ratio to 1.0, make the display with PLOTC, and then restore the aspect ratio to its PREVIOUS value.

Assembling graphical elements

PLOTC draws the elements of a display in the order they are met in the description. The coordinate system is a relative one in the plot window with (0,0) at lower left and (1,1) at upper right. Two positions are always kept track of: the *origin* and the *cursor*. Usually the origin is the previous cursor position, but it may be explicitly set. A display description consists of a series of single-letter commands and a cursor position given by two real numbers between 0 and 1. Table 7.1 lists the commands recognized by PLOTC.

Table 7.1 PLOTC commands and graphical elements

Command	Coordinates		Description
*	—	—	Comment–text ignored
Q	—	—	Draw display and quit PLOTC
O	*x*	*y*	Set origin to (*x*,*y*)
G	*x*	*y*	Set origin to (*x*,*y*) and make it global
T	*x*	*y*	Place line of text (next line in file) at (*x*,*y*)
text			
U	*x*	*y*	Place multiple lines of text (following lines in file
$text_1$			up to the next blank line) at (*x*,*y*)
$text_2$			
...			
L	*x*	*y*	Draw a line from the origin to (*x*,*y*)
A	*x*	*y*	Draw an arrow from the origin to (*x*,*y*)
C	*x*	*y*	Draw a circle centered at the origin to (*x*,*y*)
R	*x*	*y*	Draw a rectangle with opposing corners at the origin and at (*x*,*y*)
N	*x*	*y*	Draw an *N*-sided polygon centered at the origin with one vertex at (*x*,*y*)
S	*x*	*y*	Draw a sector of a circle centered at the origin with one vertex at (*x*,*y*); following command is either S or C, indicating whether the minor (S) or major arc (C) is drawn to next cursor position
B	x^a	y^a	Draw border tick marks around the plot region (if requested by the BORDER option)
M	*x*	*y*	Invoke macro name at (*x*,*y*); following line contains macro
name	*scale*	*angle*	name, multiplicative scale factor (between 0 and 1), and baseline orientation as clockwise angle from horizontal in degrees
[	—	—	Set parameters using text on the line up to closing] character

[a]Values ignored but must be present.

A PLOTC display description file typically starts by setting parameters, then sets an origin, and then draws a sequence of graphical elements, moving the origin when necessary. For example, the following file describes the display of a circle in the center of the plot with a rectangle around it:

Ex-7.4

```
* The following line sets the line style to be thicker than default
[W3]
O 0.5 0.5
C 0.5 0.8
O 0.2 0.2
R 0.8 0.8
Q
```

The commands set an origin, draw a circle, reset the origin, and draw a rectangle enclosing the circle.

Macro files are extremely useful for repeating groups of elements at different places in a display. The following commands, placed in a macro file called `ex7-1note.pcm`, draw a small cross labeled "Peak posn." when the macro is invoked.

Ex-7.5

```
* Macro; set text size to medium
[SM]
O -0.01 0.0
L  0.01 0.0
O  0.0 -0.01
L  0.0  0.01
T  0.03 0.0
Peak posn.
```

In macros, the coordinates are relative to the cursor position where the macro is used, and the coordinate axes are rotated by the angle given when the macro is invoked. This is why some of the coordinate values are negative. A Q command does not need to end a macro; the end of the file is sufficient.

The following file, `ex7-1.pcf`, creates the display and uses the macro to place text at various places in it.

Ex-7.6

```
M  0.5 0.5
ex7-1note 1.0 0
M  0.5 0.5
ex7-1note 1.0 90
M  0.5 0.5
ex7-1note 1.0 180
M  0.5 0.5
ex7-1note 1.0 270
M  0.25 0.25
ex7-1note 2.0 -45
M  0.25 0.75
ex7-1note 2.0 45
M  0.75 0.25
ex7-1note 2.0 -135
M  0.75 0.75
ex7-1note 2.0 135
Q
```

Figure 7.4 shows the resulting display. There is a cross in the center labeled with text in each 90° orientation. There are four more crosses centered in each of the four quadrants of the display, with text angled toward the center. The cross orientations are rotated along with the text and are twice as large as the center cross due to the scale factor used when the macro is used. Text is not subject to scaling (see following section).

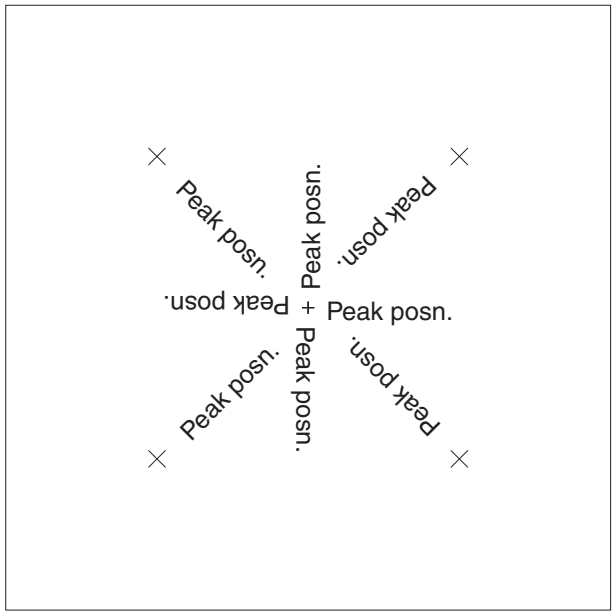


Figure 7.4 Display showing the use of a PLOTC macro in different orientations. The macro draws a cross labeled "Peak posn." The cross size depends on the scale factor used when the macro is invoked to draw the graphical elements within it. The cross and text orientation depend on the angle.

Parameters controlling graphical elements

While the shapes of graphical elements are fixed, SAC allows the way they are displayed to be changed for stylistic purposes. For example, a line might be drawn as dashed or dotted, an arrowhead might be open or filled, a rectangle might be outlined in color or black. These are called parameters by PLOTC. Parameters are set by strings enclosed in square brackets ([. . .]) that begin with the first character on a line of a display description file.

Table 7.2 lists the entire collection of parameters that may be changed in a display. Each parameter consists of a one- or two-letter code followed by an integer argument or character code. Line drawing is one area under parameter control: a line's style and width, whether symbols ornament a line and the symbols' sizes. The graphical element color is also under parameter control. Parameters also affect the way text is rendered: the font, size and justification (both horizontal and vertical). The number of polygon sides is also set by a parameter. Finally, the numbers of tick marks in the horizontal and vertical directions are also under parameter control.

7.3 USING PLOTC

SAC's PLOTC command works in two ways. Typically, it replays an existing display annotation file to annotate a data plot as part of a seismic data analysis procedure. Used this way, it typically appears in a SAC macro that implements the analysis procedure. The command is essentially replaying a pre-existing file to annotate a plot inside a frame consisting of a composite plot. Thus the command in the macro is

```
SAC> plotc replay file XXX
```

where XXX.PCF is the name of the file containing annotation commands (Fig. 7.4 was produced this way). A border is drawn around the annotated area by default. If unwanted, use the BORDER OFF command option.

Table 7.2 PLOTC parameters affecting graphical elements

Code	Argument	Description
H*j*	*j*=L,C,R	Text justification horizontal (L)eft, (C)enter, (R)ight
V*j*	*j*=B,C,T	Text justification vertical (B)ottom, (C)enter, (T)op
S*s*[a]	*s*=T,S,M,L	Text size (T)iny, (S)mall, (M)edium, (L)arge
Q*n*	*n*=1-4	Text quality 1,2=hardware, 3,4=software
Q*q*	*q*=H,S	Text (H)ardware, (S)oftware
F*n*	*n*=1-8	Text Font n
A*s*	*s*=T,S,M,L	Arrowhead size (T)iny, (S)mall, (M)edium, (L)arge
A*t*	*t*=F,U	Arrowhead type (F)illed, (U)nfilled
A*v*	*v*=V,I	Arrowhead shaft (V)isibile, (I)nvisible
L*n*	*n*=1,4	Line type 1=solid, 2-4=other
LN*n*	*n*=1,...	Line type n (can be larger than 4)
W*n*[b]	*n*=1,4	Line width n, 1=thinnest, 4=thickest
N*n*	*n*=2,9	Number of polygon sides
P*n*	*n*=1,4	No-op (ignored)
C*s*	*s*=N, other single character	Set color to N(ormal), or other color in color table identified by its first character; default color table provides R(ed), G(reen), Y(ellow), B(lue), M(agenta), C(yan), W(hite)
BH*n*	*n*=integer	Set number of horizontal border ticks to n
BV*n*	*n*=integer	Set number of vertical border ticks to n
SY*n*	*n*=integer	Set symbol number for line plotting; n=0 turns off symbols
SZ*n*	*n*=integer	Set symbol size; n in milliunits (e.g., 5 is 0.005)
	n=0	scales symbols to current character size
SP*n*	*n*=integer	Set symbol spacing; n in milliunits (e.g., 100 is 0.100)
	n=0	puts symbols at every point on line

[a]Saved and restored when a macro is invoked.
[b]WIDTH ON must be in effect for option to affect widths.

The second use of the PLOTC command is to design annotations that will become part of a data analysis procedure (or, less commonly, a one-off plot). This involves a graphical interaction with the user and a partial plot, similar to the way the PLOTPK command interacts graphically with the user and a data trace. The command to start PLOTC in this way is

```
SAC> plotc create file XXX
```

In this mode, PLOTC takes a sequence of commands and cursor positions and writes them into a display annotation file (in this case, a file called XXX.PCF). When the interaction is finished, the file contains the annotations on the plot. At this point the file can be edited with a text editor to modify the commands, cursor positions, or parameters to refine the annotations for production use.

The interaction is quite simple. When PLOTC is invoked with the CREATE option, a graphical cursor appears on the screen, and SAC waits for you to type a command character (one from Table 7.1). At that point, the command and cursor position are written to the file, and any extra information needed to describe the graphical element (the text, the macro name, etc.) is read and put into the file. SAC displays the graphical element and the cursor reappears awaiting the next command character. The interaction finishes when you type Q.

The Q character is not copied to the annotation description file. It must be added by hand using a text editor. In REPLAY mode, if the Q command is missing, PLOTC assumes that you are continuing to create an annotation and displays the graphical cursor crosshairs. If, after replaying a file, a set of crosshairs unexpectedly appears on the screen, you have forgotten to add the Q to your file.

CHAPTER

EIGHT

Array data handling

8.1 SAC SUBPROCESSES

SAC's commands potentially affect all aspects of a trace. For some types of analysis, however, it is computationally simpler to apply certain assumptions to a collection of traces before they are analyzed. For example, trace stacking is easier when it is assumed that every trace to be stacked has the same sample rate and the same start and end times. To enforce such assumptions, SAC provides the use of a *subprocess*, which, when entered, limits the commands that may be issued. SAC contains two major subprocesses: the signal stacking subprocess (SSS) and the spectral estimation subprocess (SPE). Entry to the subprocess is through the SAC commands `SSS` and `SPE`, respectively, and the return from either is through the `QUITSUB` command (abbreviated `QS`).

Inside the subprocess some commands are restricted and some other commands become available. The rest of SAC's commands continue to operate. To see which commands a subprocess allows, use `HELP COMMANDS` when inside the subprocess:

```
SAC> sss
 Signal Stacking Subprocess.
SAC/SSS> help commands

 Alphabetical list of SAC SSS commands.  Type

    help xxxx

 for information about the xxxx command, and

    syntax xxxx

 ...
```

When looking for a command using HELP APROPOS, if the command is restricted to use within a subprocess, (SPE) or (SSS) will appear after its command name.

In the previous example, note that SAC announces entry to the subprocess, and its command prompt changes. This is a reminder that use of certain commands is restricted inside the subprocess.

8.2 THE SIGNAL STACKING SUBPROCESS

The SSS command enters the subprocess, which is designed for combining many traces for graphical display, for trace picking and quality control (QC) and for stacking data to suppress noise and enhance signal. The subprocess works with a trace collection that is initially defined by the traces in memory before entering SSS. The collection may be added to or winnowed within the subprocess.

Trace collections

The trace collection (also known as a *stack*) is the set of files that SSS operates on, either to display them or to stack them. Each trace in the collection has a set of *properties* that may be defaulted or explicitly set. The properties are as follows:

- name – the file name from which the data was read;
- weight – its numerical weight when stacked with other traces;
- polarity – its polarity sense (positive/normal or negative/reversed);
- range – the position of the trace relative to an origin;
- sum – a true or false value indicating whether the trace participates in any stacks calculated with the trace collection;
- delay time and increment – the time shift relative to the trace start for alignment with other traces and an incremental time shift factor;
- sample delay and increment – the shift in samples relative to the trace start for alignment with other traces and an incremental sample shift factor;
- velocity model – a velocity model designation (1 or 2) to calculate time shifts.

The range and polarity affect plotting and picking. The weight and polarity affect stacking. The velocity model and the shifts affect trace plotting and stacking. To view the trace properties, use the LISTSTACK command.

A time window must be defined to plot or stack the trace collection. The TIMEWINDOW command defines the window, which consists of a start and an end time in seconds. The time window applies to the trace after defining all of its time shifts. There is no default time window.

A range window also must be defined to plot a trace collection. Ranges may be defined in various ways by using SAC's DISTANCEWINDOW command:

```
SAC/SSS> help distancewindow

 SUMMARY:
 Controls the distance window properties on subsequent record
 section plots.
```

```
 SYNTAX:
 DISTANCEWINDOW [{USEDATA|WIDTH <w>|FIXED <lo> <hi>}]
     [UNITS {KILOMETERS|DEGREES}]
```

The range window may be data-based (nearest and farthest trace from the origin), a fixed distance (a range from the nearest trace), or a low and high range value. Ranges may be specified in either kilometers or degrees of arc.

A trace's range value is normally its distance from an event origin. However, it can actually be any value associated with a trace. For example, the natural way to order receiver functions is on the basis of their horizontal slowness, which controls the delays of the primary conversions and the multiples from the parent (main) arrival. Thus a receiver function trace's "range" can actually be a slowness in seconds/degree or seconds/kilometer, explicitly set when the trace is added to the collection.

Adding, deleting and changing traces

The basic command to add a new trace to a collection is the ADDSTACK command:

```
SAC/SSS> help addstack

 SUMMARY:
 Add a new file to the stack file list.

 SYNTAX:
 ADDSTACK <filename> [WEIGHT <v>] [DISTANCE <v>]
     [DELAY <v> [{SECONDS|POINTS}]]
     [INCREMENT <v> [{SECONDS|POINTS}]]
     [{NORMAL|REVERSED}] [SUM {ON|OFF}]
```

This command names a file to be added to the trace collection. All other options act to define the trace's properties. If a property is not defined, it inherits a default property set by the GLOBALSTACK command. If the DISTANCE property is neither defined nor a global default specified, it is taken from the event and station information (EVLA, EVLO, STLA, STLO) if known.

The CHANGESTACK command changes trace collection information:

```
SAC/SSS> help changestack

 SUMMARY:
 Change properties of files currently in the stack file list.

 SYNTAX:
 CHANGESTACK {<name>|<number>|ALL} [WEIGHT <v>] [DISTANCE <v>]
     [DELAY <v> {SECONDS|POINTS}]
     [INCREMENT <v> {SECONDS|POINTS}]
     [{NORMAL|REVERSED}] [SUM {ON|OFF}]
```

A trace may be designated either by its name, by its numerical position in the collection, or by the whole set of traces "ALL". Similarly, a trace may be purged from the collection by using the DELETESTACK command:

```
SAC/SSS> help deletestack

 SUMMARY:
 Deletes one or more files from the stack file list.

 SYNTAX:
 DELETESTACK <name>|<n> ...
```

Plotting record sections

After a collection of traces is defined, they may be plotted by using the PLOTRECORDSECTION command (abbreviated PRS). A basic record section plot of some of SAC's built-in data is shown in Figure 8.1.

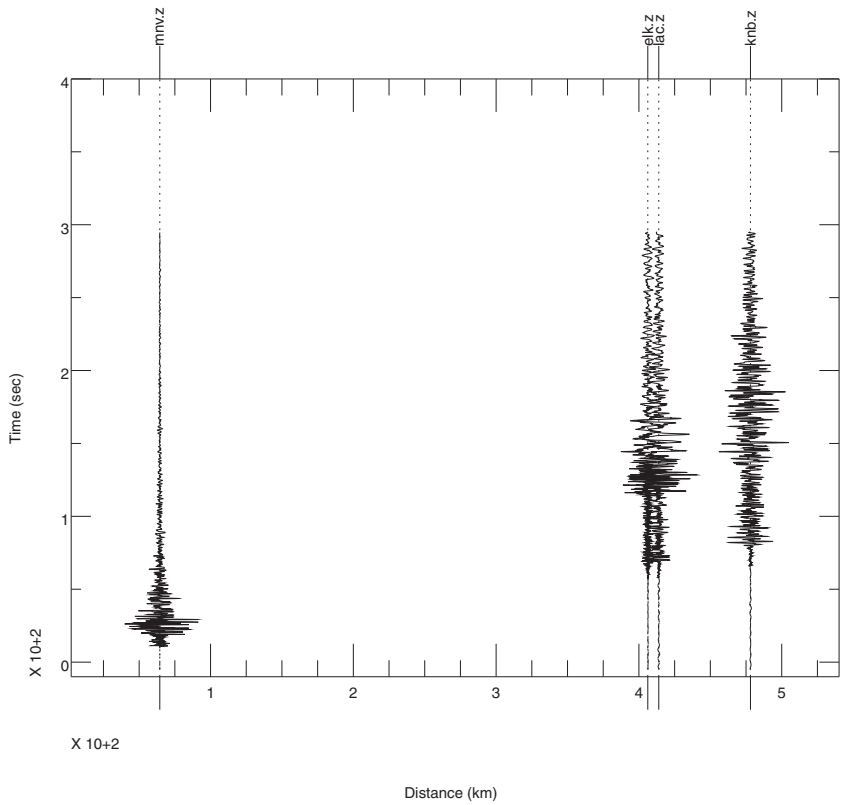


Figure 8.1 This plot shows four vertical component seismogram records from a single event using PLOTRECORDSECTION. The X axis is the range from the event origin in kilometers. The Y axis is the time in seconds from the reference time. Each trace is plotted relative to an amplitude axis anchored at the distance of the station from the range reference (in this case, the event location). The axis is shown with a dotted line and labeled at the top of the figure with the file name. With increasing distance, the P-wave arrival is later in the record section, as expected from normal crustal travel times.

Ex-8.1

```
SAC> datagen sub regional *.z  ;* Built-in data
SAC> sss
SAC/SSS> tw -10 400            ;* Define time window
SAC/SSS> prs                   ;* Default record section plot
```

This is a default plot. Each trace in the collection is plotted with a baseline (zero level) at its range from the reference position. The baseline position is always marked outside of the axis frame by a tab that plays a role in picking traces (see Section 8.2). PLOTRECORDSECTION (abbreviated PRS) has many options to change plot details. Of particular note is the ability to label traces (see the LABEL option) with information other than the file name; any file header variable value may be used as a label. The reference line (the zero amplitude range position) may also be suppressed (see REFERENCELINE). Finally, the orientation of the plot may be changed from landscape (a horizontal range axis) to portrait (a vertical range axis). See the ORIENTATION and ORIGIN options for axis orientation.

Record section plots may be annotated with any of the features described in Chapter 7. It is often desirable to see travel-time curves for particular arrivals from an earthquake. The SSS subprocess provides a simple way to define which travel-time curves to plot using the TRAVELTIME command. The example below shows how to produce a simple record section plot with a set of P and S travel times. Figure 8.2 shows the resulting plot:

Ex-8.2

```
SAC> datagen sub regional *.z         ;* Built-in data
SAC> sss
SAC/SSS> traveltime model iasp91 phase Pn Sn
SAC/SSS> tw -10 400                   ;* Define time window
SAC/SSS> prs ttime on orient portrait ;* Portrait, travel times
```

Here, the TRAVELTIME command requests that travel-time curves for the regional P and S arrivals (Pn and Sn) be calculated. The PRS TTIME ON option adds the curves to the record section, and the ORIENT PORTRAIT option changes the distance axis to vertical. If you do not like the sense of increase along the range axis (left-to-right in LANDSCAPE orientation, top-to-bottom in PORTRAIT), use the ORIGIN REVERSED option to change it.

Stacking

SAC's SSS subprocess also provides the ability to stack the traces in the collection. The idea underlying stacking is that combining many traces recording the same arrival in a seismic wavefield suppresses noise. The noise is presumably incoherent, and therefore the noisy parts of the traces will sum to a value whose average is zero. Only the coherent parts of the traces will survive the summation to yield a nonzero value. In this way, signal is enhanced over noise.

The SUMSTACK command implements stacking. A collection of traces must be defined along with a time window (see the TIMEWINDOW command). The data in the time window is summed after applying all the time delays associated with the trace properties.

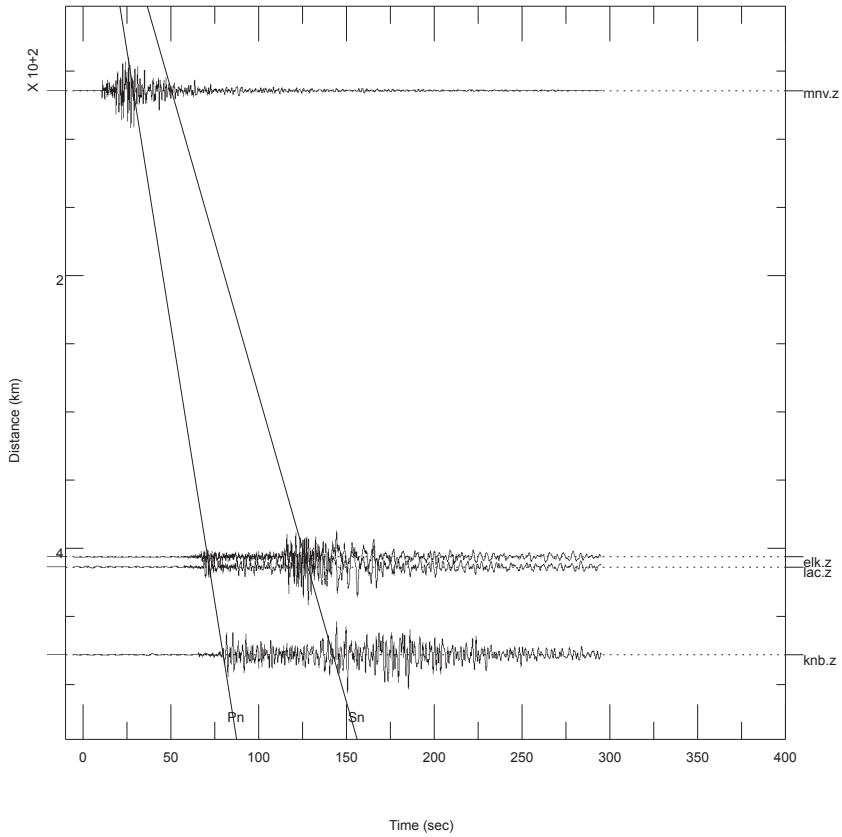


Figure 8.2 This plot is similar to the previous record section plot but differs in the orientation of the range axis (vertical) and added travel-time curves. The regional P- and S-wave arrivals match the first disturbance (Pn) and the largest amplitude arrival (Sn) on almost all of the traces except the closest one.

By default, there is no time delay other than the one defined by the ADDSTACK or CHANGESTACK command. However, further time delays may be specified using the VELOCITYMODEL command.

Velocity models

SAC's VELOCITYMODEL command (abbreviated VM) recognizes two types of velocity models: one for a direct arrival (a so-called "refracted wave" model) and another for a reflected arrival (from a horizontal discontinuity like the Moho, called a "normal moveout" model). The key function of a velocity model is to provide a way to adjust trace delays depending on distance. Adjustments made to the trace delays sharpen waveform features, if the arrival in the stack behaves like one of the models. INCREMENTSTACK successively adjusts the trace delays through increments defined by either the VELOCITYMODEL command or the ADDSTACK or CHANGESTACK commands. INCREMENTSTACK behaves like the stacking control parameter (the relative slowness) in a slant stack.

The refracted wave model defines a time delay t_{delay} as follows:

$$t_{delay} = t_{R,VM} - (t_{0,VM} + \Delta / V_{app}) \tag{8.1}$$

This identifies a lag due to a direct wave propagating a distance Δ (km) divided by an apparent velocity V_{app}. There are two lags, one used to correct for a particular source depth (t_0) and one for alignment at a reference range (t_R). These are set with the `VM T0VM` and `VM TVM` options, respectively, and V_{app} with `VM VAPP`.

The reflected wave model defines a time delay t_{delay} as follows:

$$t_{delay} = t_{R,VM} - \sqrt{t_{0,VM}^2 + (\Delta / V)^2} \tag{8.2}$$

This identifies a lag due to a wave propagating downward from a source and reflecting off a horizontal interface at some level. In this case the delay is a hyperbolic function that consists of a constant factor t_0 representing the arrival time for a reflection from the interface at zero lag (depth/velocity) and a range-dependent factor (Δ/velocity) depending on the constant velocity V in the layer above the interface. An alignment time at a reference range (t_R) is included. These times are set with the `VM T0VM` and `VM TVM` options, respectively, and the velocity is set with `VM VAPP`.

To shift the timings in the stack, increments may be defined for any (or all) t_0 and V_{app} or V. The reference timings t_R are defined once per model with `VM TVM` at the reference range given by `VM DVM`. SAC increments stack control values each time the `INCREMENTSTACK` (abbreviation `IS`) command is used. Use `LISTSTACK` to see the prevailing delays resulting from application of the velocity model. Usually only one item is incremented at a time so that its effect may be understood. In practice, the most common use of velocity models is to use none at all – the static trace delay is enough to define the lags used for trace alignment.

Types of stacks

The goal of stacking is to enhance signal by suppressing noise. The straightforward way to stack is simply to sum each sample at the corresponding time in each trace and to divide each sample sum by the number of traces. This results in a *linear stack*.

A slightly more sophisticated way is to recognize that a signal typically lies at a higher amplitude than noise. By nonlinearly stretching the amplitude axis, the amplitude range is made more nearly constant. Thus the incoherent noise is suppressed more readily through its random phase rather than its lower amplitude relative to the signal (McFadden *et al.*, 1986). After summation, the resulting sum is nonlinearly shrunk back to its original amplitude range. The nonlinear stretching is usually done by taking the Nth root of the sample value, where N is usually a small positive integer. After calculating the sum, the result is raised to the Nth power. This method is called the *Nth root stack* due to the nonlinear mapping.

Another method that recognizes the importance of phase coherence in separating signal from noise is phase-weighted stacking (Schimmel and Paulssen, 1997). This method explicitly computes the phase of each trace point from the analytic signal and weights the point by its phase sum. A coherent phase sum of M point samples has a magnitude of M whereas an incoherent phase sum has a magnitude close to zero. The amplitude sum at each point is therefore multiplied by the Nth power of the phase sum, which suppresses incoherent samples in the stack. It is called a *phase-weighted stack* on this account.

No matter which stacking method is chosen, traces with high signal amplitudes will be emphasized over those with lower amplitudes. Thus the traces can be normalized by their trace weights. If the weight is taken as the reciprocal of the peak trace amplitude, the

overemphasis of high-amplitude traces is ameliorated. Use ADDSTACK to provide suitable trace weights.

Stack uncertainty bounds

It is useful to have an estimate of the uncertainty associated with each sample point in a stack. For a linear stack, this may be defined easily through the variance of each sample point and its relationship to a confidence level through the χ^2 distribution. For the other stacking methods, this concept is no longer useful because the variance is of a sum of a distorted sample and the statistical model no longer connects it to a confidence level. Fortunately, jackknife uncertainties may be used to calculate uncertainty in this situation (Efron, 1982). The method calculates uncertainty by forming M subsets of the M traces by leaving out each trace successively. The variance of the sample estimates from each of these subsets of size $M-1$ may be used to estimate the uncertainty. The price of this approach is that when many traces are in a stack it can be slow to calculate.

SAC retains the uncertainty bounds for the stack along with the stack in memory and will display it along with the summation trace using the SUMSTACK command. It will also write it out to a file using WRITESTACK.

Example: 2006 North Korea nuclear test

Data recorded by the Japanese Hi-net array (Obara *et al.*, 2005) of the 9 October 2006 North Korean nuclear test provide a good example of the use of the SSS stacking commands. The following commands download the dataset and show a record section plot:

Ex-8.3

```
SAC> read ds http://www1.gly.bris.ac.uk/MacSAC/korea.sds
SAC> sc curl http://www1.gly.bris.ac.uk/MacSAC/korea.mac \
   > /tmp/korea.mac
SAC> sc awk '/sss/,/quitsub/{print}' /tmp/korea.mac ;* List file
sss
timewindow -5 12
distancewindow fixed 6 12.5 units degrees
changestack flt/KSIH.U delay -123.115800 s n w 1.833053
changestack flt/HIRH.U delay -123.310400 s n w 1.049097
changestack flt/MHSH.U delay -124.211400 s n w 2.782429
...
changestack flt/HNKH.U delay -203.844500 s n w 1.500755
message " 753 traces."
border off
beginframe
plotrecordsection referenceline off labels off size 1 weight off
plotc replay file /tmp/tmp_rs
endframe
border previous
quitsub
SAC> m /tmp/korea.mac               ;* Produces annotated plot
```

The first command reads a collection of SAC files in dataset format from an internet source. The second command uses the UNIX *curl* command to read a SAC macro from the

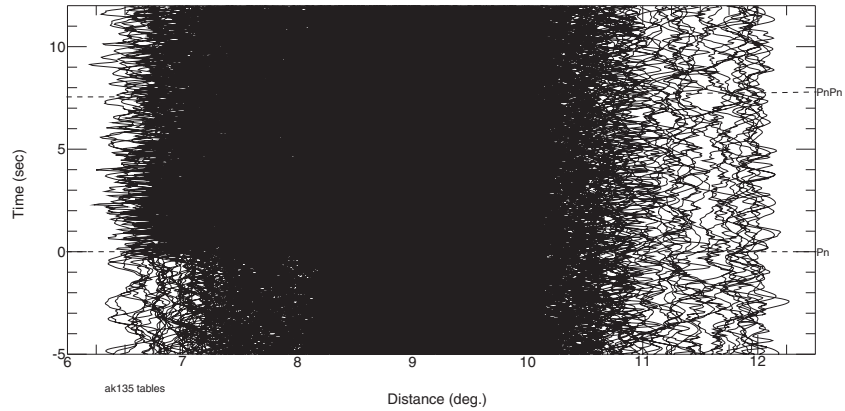


Figure 8.3 This plot shows Japanese network recordings of the 9 October 2006 North Korea nuclear test. The traces, roughly aligned on the predicted Pn arrival from the explosion (horizontal dashed line at relative time 0), do not show a signal that is significantly above the noise level. Its presence is evident, however, by a change in frequency between 6.5 and 10.5°.

same internet source. This file is a SAC macro and contains plot annotation commands followed by SSS commands to align the traces to the Pn arrival from the event. Only the part of the file containing SSS commands is shown (and significantly shortened for this example).

First, a time window is defined using `TIMEWINDOW` and a distance window by `DISTANCEWINDOW`. The `READ DS` command read the traces into memory, so after entering `SSS`, the traces are already in the collection. However, their properties need to be adjusted for alignment purposes. The series of `CHANGESTACK` commands sets the lag for each trace in seconds, sets the polarity to `NORMAL` (`N`), and sets a weight related to the maximum trace amplitude. Finally, after all the trace properties are set, the macro writes a message giving the number of traces in memory.

A series of plotting commands follow. The border around the plot is temporarily suppressed (the `BORDER OFF` ... `BORDER PREVIOUS` pair) and a composite plot constructed (`BEGINFRAME` ... `ENDFRAME`) consisting of a record section (`PLOTRECORDSECTION`) followed by annotation of the record section with the expected Pn arrival time (`PLOTC REPLAY`). For brevity, the macro commands to set up the `PLOTC` annotation file (`/tmp/tmp_rs.pcf`) are omitted here.

The basic record section plot of the data is shown in Figure 8.3. The explosion arrival is not significantly visible above the noise level. Stacking can enhance the signal, however. The commands

Ex-8.4

```
SAC> sss
SAC> cs all sum on              ;* Add all traces to stack sum
SAC> sumstack type linear
SAC> sumstack type nthroot 2
SAC> sumstack type phaseweight 2
SAC> writestack /tmp/pw-sum.sac unc /tmp/pw-unc.sac ;* Save
```

make a series of stacks with different methodologies, resulting in the plots shown in Figure 8.4. The linear stack is not nearly as good in suppressing noise as is the Nth root or the phase-weighted stack.

Saving stack data and uncertainties

SUMSTACK produces both a stacked signal estimate and the uncertainty of that estimate. The previous example showed the stack and its uncertainty written into the files /tmp/pw-sum.sac and /tmp/pw-unc.sac. To retrieve them as traces in their own right, use an approach as shown in the following commands:

Ex-8.5

```
SAC> read /tmp/pw-unc.sac /tmp/pw-unc.sac /tmp/pw-unc.sac
SAC> mul 1 0 -1          ;* pos., zero and neg. uncertainty
SAC> addf /tmp/pw-sum.sac /tmp/pw-sum.sac /tmp/pw-sum.sac
SAC> ch file 1 kevnm +1_sigma      ;* name traces for P2
SAC> ch file 2 kevnm stack_sum
SAC> ch file 3 kevnm -1_sigma
SAC> fileid type list kevnm        ;* KEVNM identifies trace
SAC> line list 5 1 5 increment on  ;* unc traces dashed
SAC> p2                  ;* overlay three plot traces
```

This reads three copies of the uncertainty and then turns them $+1\sigma$, 0σ, and -1σ values, to which the stack estimate is added. The KEVNM file header identifies the trace's identity, and the FILEID command tells SAC to use it to label the trace in future plots. The PLOT2 command yields a composite plot that shows the $\pm 1\sigma$ envelope around the signal estimate (Fig. 8.5).

Picking data in stacks

SAC's PLOTRECORDSECTION command also includes an interactive mode, like PLOTPK, where actions are controlled by the screen's cursor and keyboard. The features provided in this interactive mode include the following:

- zooming in on parts of the record section (either in time or in range);
- amplifying and shrinking traces;
- estimating the apparent velocity of features on the trace;
- selecting traces for further analysis;
- picking trace features (peaks, troughs, or zero crossings).

The PLOTRECORDSECTION CURSOR command enters interactive mode. It draws a record section plot and displays crosshairs on the screen to prompt for graphical interaction. As with PPK, a keyboard character indicates an action to take. Table 8.1 shows a complete list of possible actions. Some actions require use of macro language features to be exploited because they return values in blackboard variables.

The basic action is windowing the trace display. The trace amplitudes may be expanded or shrunk using the + or - keys. A pair of X key presses defines a zoom into a distance

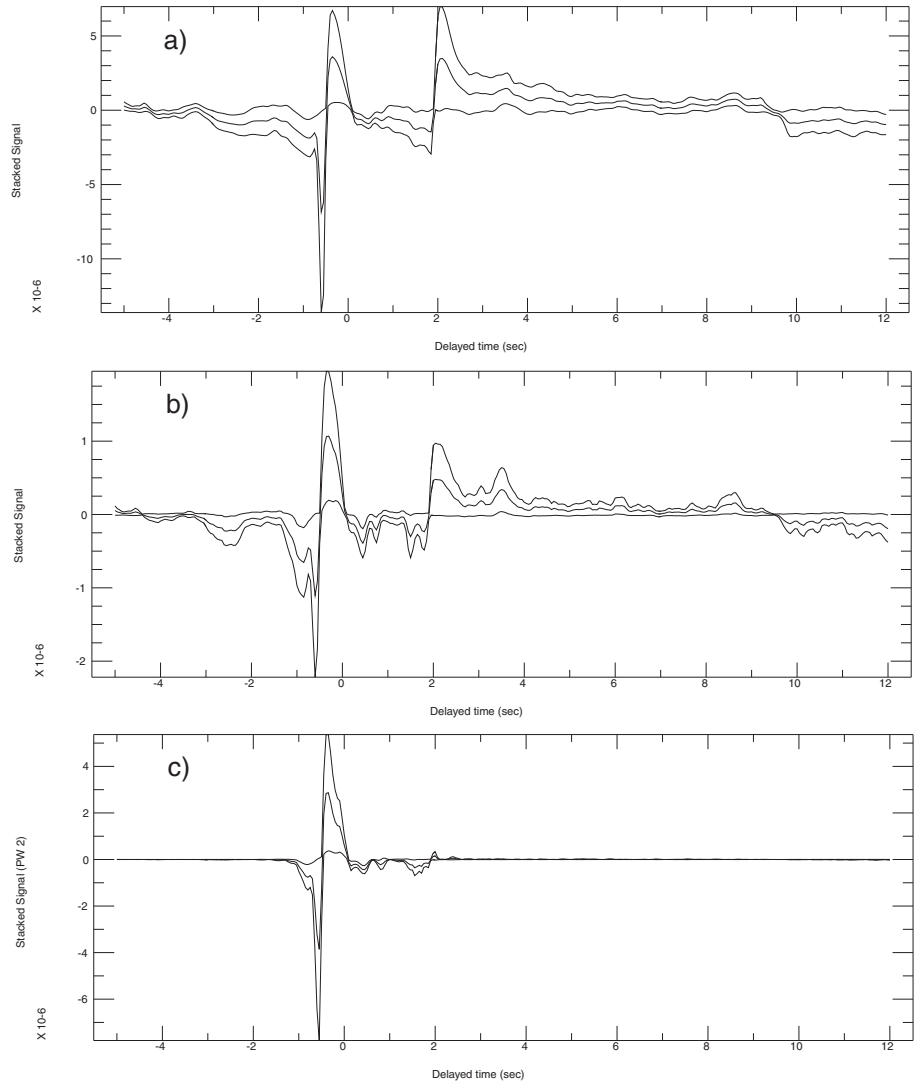


Figure 8.4 This plot shows SAC's three stacking methods applied to Japanese network recordings of the 9 October 2006 North Korea nuclear test. The traces are roughly aligned on the predicted Pn arrival from the explosion. (*a*) Linear stack, (*b*) *N*th root stack ($N = 2$), (*c*) phase-weighted stack ($N = 2$). `SUMSTACK` produced each of the plots, which consists of a central trace (the stacked estimate) and the $\pm 1\sigma$ jackknife uncertainty on each sample. Note the higher pre-arrival noise levels in the linear stack and the nearly total suppression of the incoherent second arrival at 2 s by the phase-weighted stack. The later arrival is probably due to trace misalignment rather than any property of the source.

Table 8.1 `PLOTRECORDSECTION` cursor operations

Char.	Description
K	Finish PPK
Q	Finish PPK
X	Define bound of new range window (low then high)
C	Define bound of new crop window (range and time)
O	Return to last range/crop window (the five most recent windows are kept)
R	Redraw current window, clearing messages
S	Select all traces in a distance range (first low range, then high). Sets blackboard variables[a] SSSSELECTED and SSSUNSELECTED
T	Toggle selected/unselected status of trace
+	Increase trace sizes by a factor of 2^b
-	Decrease trace sizes by a factor of 2^b
M	Define point on a moveout curve; displays line fixed to cursor leading from previous point on curve. X and O allowed while moveout curve being defined
U	Undo previous point on moveout curve
V	Report slope of moveout curve (s/km or s/deg). Sets blackboard variable VAPP
P	Pick points along moveout curve (at peak P+, trough P-, zero P0). Sets blackboard variable[a] SSSPICKS

[a]Blackboard variable is set or removed after each use of `PLOTRECORDSECTION`.
[b]Redraws window.

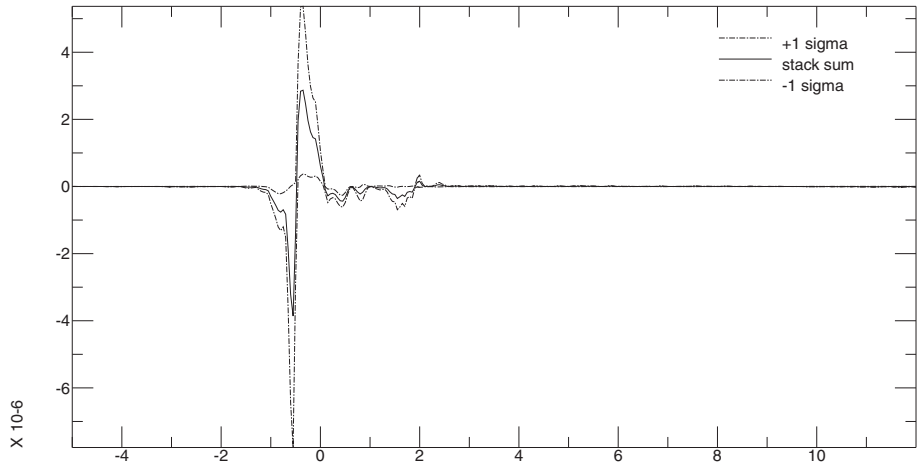


Figure 8.5 Stacked trace from an $N = 2$ phase-weighted stack, saved by `WRITESTACK`, re-read and displayed by `PLOT2`. The stack is shown along with its $\pm 1\sigma$ jackknife uncertainty.

window to expand a set of traces. The O key unzooms to the previous window. Whereas only the cursor's range position is relevant when the X key is used, the C key defines a crop of the data in both time and range. The cursor's position defines the lower left corner of the crop window at the first C key press, and the upper right corner is defined by the second.

Only the selected time and distance window is displayed following the crop. The O key also unzooms to the previous window. R redraws the display.

Trace features may be measured and recorded by using the M key to define anchor points of a moveout curve. Every time the M key is pressed, it defines a new point on the moveout curve. A line leads from the previous moveout curve point to the cursor's position. This defines a slope (in seconds/degree or seconds/kilometer depending on the range unit) related to an apparent velocity of a feature recorded in traces at different ranges. Each time the V key is pressed, SAC reports the slope of the last segment of the moveout curve on the screen (R will erase previous slopes and redraw the screen). The X and C commands may be used to zoom in on portions of the record section even while defining a moveout curve. The moveout curve stays active until `PLOTRECORDSECTION` finishes or until a pick is made.

The points where the moveout curve crosses the trace baselines in the plot provide a time reference where picks on each trace may be made. When a moveout curve is active, the commands P+, P- and P0 cause a pick to be made on every trace crossed by the moveout curve. The nearest peak (+), trough (-) or zero (0) crossing to the intersection defines the pick. SAC saves the pick values in the blackboard variable called SSSPICKS as (trace number, time) pairs. These may be read and handled using the `WHILE READ` command in a loop to act on each pick: write it to a file, change the properties of the trace in the stack, etc.

Groups of traces may also be selected graphically. The range position of the cursor defines the beginning or end of a group of traces after S is typed. To flag the selected traces, the trace's baseline tab changes to the color red. Individual traces also may be selected or deselected by using the T character to toggle its selection status. Similar to the way that trace pick information is returned, SAC sets the trace numbers of the selected traces in the SSSSELECTED blackboard variable, and the trace numbers of the unselected traces in the SSSUNSELECTED blackboard variable. The trace numbers may be read and processed in a loop in a macro using the `WHILE READ` command.

The SSSPICKS, SSSSELECTED and SSSUNSELECTED blackboard variables are set and erased after each `PLOTRECORDSECTION` command. If relevant, they must be processed or saved before viewing the next record section. Use *quotebb* when saving these values to preserve the (number, time pick) association.

Picking examples

The following command examples illustrate the ideas associated with trace selection and time picks in record sections using some of SAC's built-in data. The `DATAGEN SUB LOCAL` dataset is a collection of trace records of a local event. The data contain a few bad traces that will be discarded after visual selection.

Ex-8.6

```
SAC> sc cat ex-prs-remove.m
* Contents of file ex-prs-remove.m
if Y EQ (existbb SSSSELECTED)
   while read SSSSELECTED n
      message "Removing &$n$,filename&"
   enddo
   deletestack %SSSSELECTED%
else
   message "No traces selected for removal"
```

```
endif
SAC> datagen sub local *.z; rmean ;* Remove baseline offset
cal.z cao.z cda.z cdv.z cmn.z cps.z cva.z cvl.z cvy.z
SAC> sss
SAC/SSS> tw 10 50; dw usedata
SAC/SSS> prs c                 ;* Use T to select bad traces
SAC/SSS> m ex-prs-remove.m
 Removing cda.z
 Removing cps.z
SAC/SSS> prs                   ;* Redisplay without bad traces
```

The final example involves making picks using a moveout curve. The P-wave arrival in the same collection of traces is clear and has an apparent velocity that causes it to appear later in time as range increases. In order to change the static shifts of each trace so that the P arrival appears at zero relative time in the window, use the pick option to approximately align to the P-wave peak. Then, adjust the static delay for each trace to remove the moveout with range.

Ex-8.7

```
SAC> sc cat ex-prs-align.m
* Contents of file ex-prs-align.m
if Y EQ (existbb SSSPICKS)
   while read SSSPICKS n t
      message "Offset for &$n$,filename& changed to $t$"
      changestack $n$ delay -$t$ s
   enddo
else
   message "No picks made"
endif
SAC> datagen sub local *.z; rmean
cal.z cao.z cda.z cdv.z cmn.z cps.z cva.z cvl.z cvy.z
SAC> sss
SAC/SSS> tw 10 50; dw usedata
SAC/SSS> prs c                ;* Use M and P+ for picks
SAC/SSS> m ex-prs-align.m
 Offset for cal.z changed to 20.848566
 Offset for cao.z changed to 23.187908
 Offset for cda.z changed to 23.281334
 Offset for cdv.z changed to 19.336441
 Offset for cmn.z changed to 20.839802
 Offset for cps.z changed to 22.320883
 Offset for cva.z changed to 20.935202
 Offset for cvl.z changed to 22.021576
 Offset for cvy.z changed to 22.235571
SAC/SSS> tw -10 40           ;* New reference time
SAC/SSS> prs                 ;* Aligned to P-wave arrivals
```

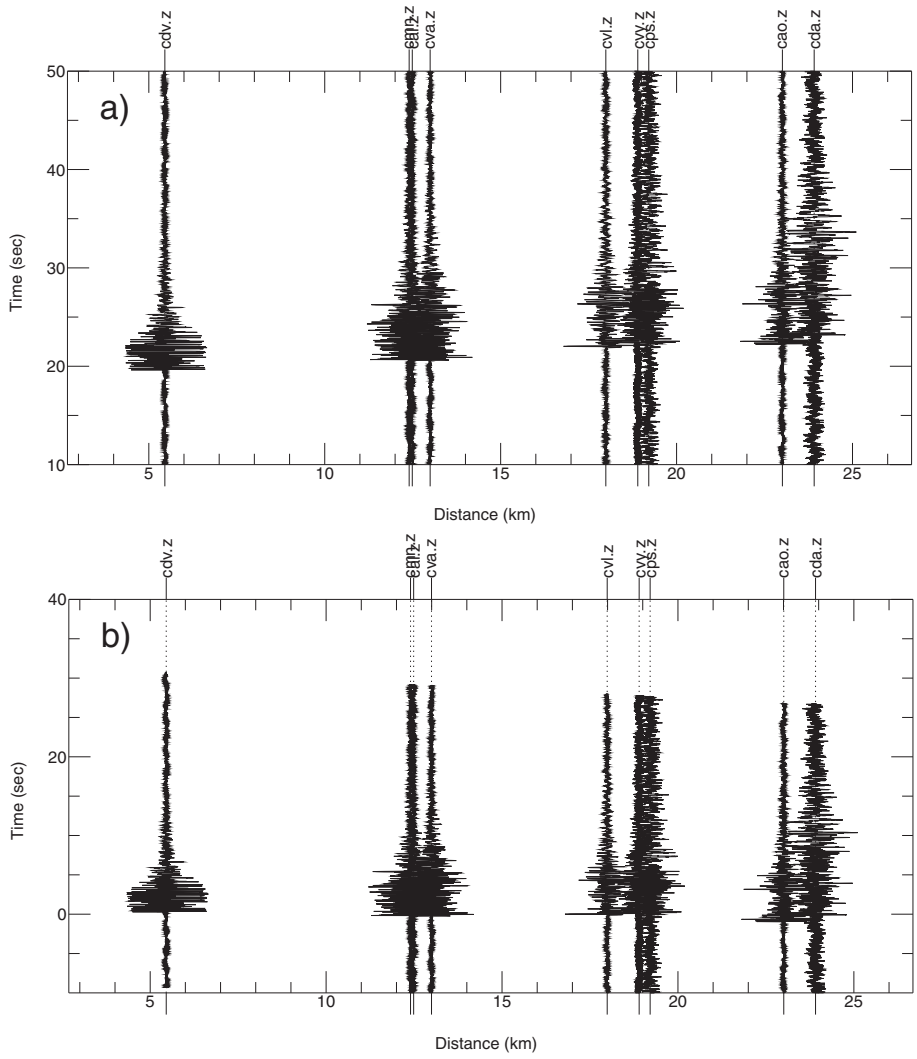


Figure 8.6 Vertical component trace collection of a local event. Time zero is the origin time of the event. Thus, with the default static delays, the P-wave arrival is later with increasing range (*a*). After making a set of picks along a moveout curve, the static delays for each trace are adjusted to bring the P arrival to relative time zero in each trace (*b*). The adjustment is made automatically by a macro processing the picks retained in SAC's memory.

Figure 8.6 shows the trace configuration before and after the picking defines the new static offsets for each trace. The first use of `PRS` involves a graphical interaction. Using M to define a moveout curve extending from (20 s, 5 km) to (22 s, 25 km), and then using P+ to make picks, the offsets are available for the macro `ex-prs-align.m` to make new static shifts. The second record section plot shows the P arrivals aligned at relative time zero in the time window.

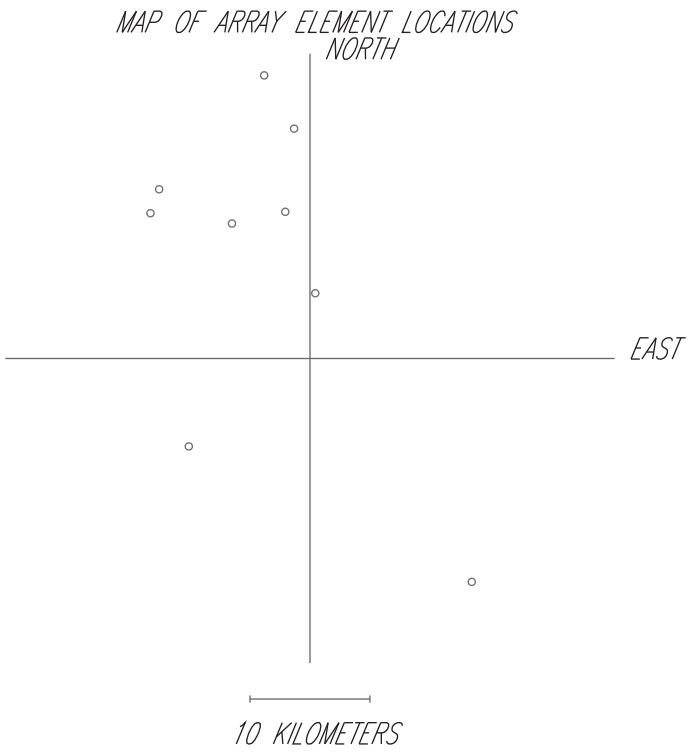


Figure 8.7 This plot shows an X–Y map of stations in a local network recording an earthquake, produced by ARRAYMAP. The origin of the map is the epicenter, and circles show individual station locations. (For color version, see Plates section.)

8.3 ARRAY MAPS

Whenever a trace collection is read in, the geometry of the stations may be relevant. For example, the azimuthal coverage around the event origin might be useful to assess how well the event is located, because gaps along particular azimuths lead to increased epicentral uncertainty in that direction. Another example is: when a regional velocity model is to be derived from the moveout of an arrival recorded by a regional network, path coverage knowledge would help assess the model's applicability.

SAC provides a quick way to make the assessments by producing a station map evoked by the ARRAYMAP command. SAC makes a local Cartesian approximation based on either the first station's location or a fixed point given by a CENTER latitude and longitude and calculates a range and azimuth to every other station that is turned into an (X,Y) coordinate. The commands below produce a map (Fig. 8.7) of a local network centered on the event epicenter:

Ex-8.8

```
SAC> datagen sub local *.z
SAC> arraymap center &1,evla& &1,evlo&
```

Array maps are also useful for assessing the geometry of small-aperture arrays used for nuclear discrimination purposes. ARRAYMAP COARRAY will plot the array symmetries that define its angular resolving power.

8.4 BEAMFORMING

BBFK and BEAM are two commands that SAC provides to do array analyses of the approach of a signal that is modeled as a plane wave. The plane wave assumption is clearly better when the array is small-aperture, but over teleseismic distances the approximation holds even for regional seismic arrays (e.g., the national seismic networks of Japan and parts of USArray).

BBFK (broadband f–k) is designed to determine the bearing of approach of a signal toward the array. Under the signal's plane wave approximation, it has a horizontal slowness p. The wavenumber k is related to frequency f and slowness via

$$k = f \times p \tag{8.3}$$

If f is a frequency in Hz and p a slowness in s/km, then k is 1/km. The signal is assumed to be spread across the wavenumber spectrum, which is equivalent to saying that it is a polychromatic signal at a single slowness p (a good model for a seismic wave arrival) or that it is a monochromatic signal at a range of slownesses (unlikely in the seismic case). If an array of sensors is spread over the ground, both the array geometry and the assumption of a wideband signal help to break the spatial aliasing inherent in the approach. In particular, the wideband assumption means that the wavenumber spectrum, plotted in polar form with the angle representing approach bearing, will radiate in a ridge from zero wavenumber (the center) toward the outer circumference of the plot. Spatial aliasing due to the array will also create ridges parallel (in a Cartesian sense) to the true spectral bearing but at different offsets from the center. Once it is identified, the peak amplitude along the ridge will show the dominant wavenumber of the signal. The horizontal slowness may be calculated from Equation 8.3.

A synthetic dataset may be used to illustrate the ideas, based on the macro listed below:

Ex-8.9

```
SAC> sc cat ex-bbfk.m
* BBFK example macro. Usage: m ex-bbfk.m [zero | n-s | e-w | diag]
$default 1 zero
setbb cutew * cutns *             ;* no cuts by default
if $1$ eq diag
   setbb cutew cutim cutns cutim  ;* time shifts N-S and E-W dirs
endif
if $1$ eq n-s
   setbb cutns cutim              ;* time shift on N-S dir
endif
if $1$ eq e-w
   setbb cutew cutim              ;* time shift on E-W dir
endif
cuterr fillz                      ;* Zero exterior trace areas
setbb xo -1 yo 0                  ;* xo and yo are x-y offsets
do f list a b c d e f g h         ;* E-W array arm
   fg seismogram; rmean           ;* Sample data
   %cutew% (&1,b& - %xo%) (&1,e - %xo%)      ;* shift: add/cut 1 s
   ch kuser1 &1,kstnm& user7 %xo% user8 %yo% ;* element coord.
   write /tmp/arr_r$f$.sac                   ;* write data file
```

```
   setbb xo (before . (%xo% + 1))            ;* increment X
enddo
setbb xo 0 yo -1                  ;* Reset offsets
do f list a b c d e f g h         ;* N-S array arm
   fg seismogram; rmean           ;* Sample data
   %cutns% (&1,b& - %yo%) (&1,e - %yo%)      ;* shift: add/cut 1 s
   ch kuser1 &1,kstnm& user7 %xo% user8 %yo% ;* element coord.
   write /tmp/arr_b$f$.sac
   setbb yo (before . (%yo% + 1))            ;* increment Y
enddo
sc rm /tmp/arr_rb.sac             ;* Repeated array element
read /tmp/arr_[r,b]*.sac          ;* Read traces
bbfk filter off wave 1 pds norm size 180 100
SAC> m ex-bbfk.m zero
SAC> m ex-bbfk.m n-s
SAC> m ex-bbfk.m e-w
SAC> m ex-bbfk.m diag
SAC> arraymap
```

The synthetic data is a built-in seismogram that is replicated at each station in the array with time shifts of 1 s related to either or both of its horizontal and vertical positions. Thus the apparent horizontal velocity is 1 km/s N–S, E–W or 0.707 km/s at 45°. The array form is that of the UK Atomic Energy Agency (UKAEA) arrays, a rectilinear cross with non-symmetric arms (Rost and Thomas, 2002) (Figure 8.8). `CUTERR FILLZ` adds 1 s to the front or cuts 1 s from the end of the synthetic data trace for every kilometer of offset along the array arms. The `CUTIM` commands either act on the trace or are comments (depending on the macro parameter), being expanded through the `cutew` and `cutns` blackboard variable values, set with `SETBB`. `BBFK` expects array offsets to be set in the `USER7` and `USER8` file header variables for small-aperture arrays. Otherwise they are taken from the `STLA` and `STLO` location of the station.

Figure 8.9 shows the results of the BBFK analysis on the synthetic data. With no delays, the signal (plane wave) appears to be emerging vertically and arriving at each sensor simultaneously. The peak is at zero wavenumber (infinite horizontal velocity), and the response function of the array (Aki and Richards, 1980) is evident in the aliasing pattern due to the element separation (Fig. 8.8). With delays along the N–S array arm, the signal appears as a wideband ridge extending N–S. The peak is at k = 0.12121 km^{-1}. We know p = 1 s/km, so the signal's dominant f must be 8.25 Hz, which an inspection of the spectrum bears out.

The diagonal approach angle shows the aliasing of the signal most clearly. The ridge extending from the origin is the true plane wave representation. The others are aliases of it. With these observations in hand, BBFK analysis of a real signal may be better appreciated.

The built-in `DATAGEN SUB ARRAY` dataset contains regional array records of a teleseismic arrival. The record section plot in Figure 8.10, constructed using the commands below, shows that the core phases PKPdf, PKPbc and PKPab are fairly clearly recorded by the network. The moveout across the array of the core arrivals (PKPdf horizontal slowness is approximately 1.6 s/deg, PKPbc 2.5 s/deg and PKPab 4.1 s/deg). From PKPab we can estimate that the largest expected wavenumber k will be 0.037 km^{-1}. Therefore, the maximum

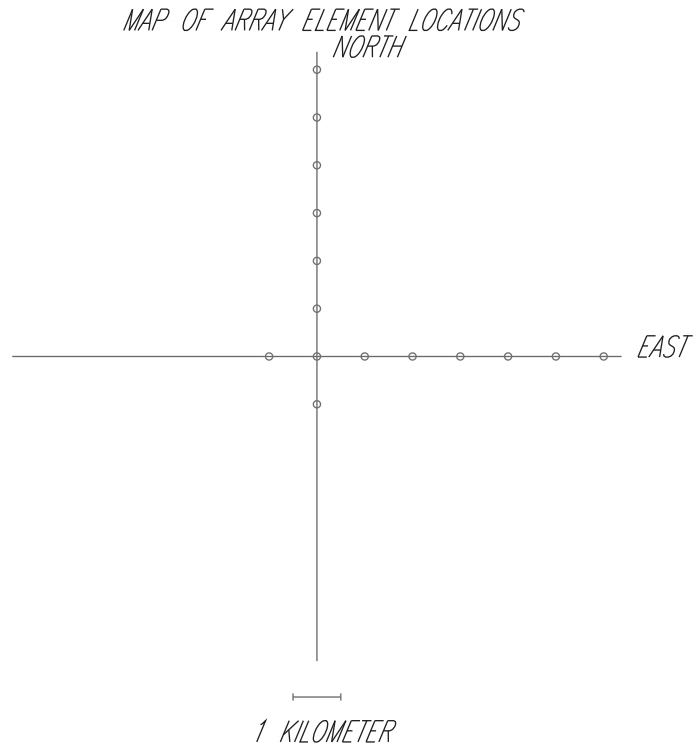


Figure 8.8 This plot shows an X–Y map of the example local array whose synthetic response is explored by the `ex-bbfk.m` macro (see text). The plot is produced by `ARRAYMAP`. Elements are spaced every kilometer along two asymmetric arms, similar to the UK Atomic Energy Agency design. (For color version, see Plates section.)

k to be explored is 0.045 km^{-1}. `BBFK` retains the peak position in blackboard variables BBFK_WVNBR and BBFK_BAZIM.

Ex-8.10

```
SAC> datagen sub array *.SHZ
SAC> sss
SAC/SSS> dw usedata units deg; tw 1100 1170
SAC/SSS> traveltime model ak135 phase PKPbc PKPdf PKiKP PKPab
SAC/SSS> line fill yellow/0
SAC/SSS> prs ttime on label kstnm
SAC/SSS> qs
SAC> bbfk wave 0.045 size 180 100 exp 4 ;* Finer az. sampling
SAC> message "k %BBFK_WVNBR%, az %BBFK_BAZIM%"
```

The result is shown in Figure 8.11. The contoured power shows a peak at k = 0.0159 km^{-1} along azimuth 355°. Using `SUMSTACK` and `WRITESTACK`, a stack of the traces aligned on PKPbc arrival indicates that its spectral peak lies at 0.7 Hz. At this frequency, the peak wavenumber indicates that the slowness at peak power is p = 0.0227 s/km or 2.526 s/deg. This corresponds to the PKPbc slowness expected for this distance range, which is the major arrival in the record section (Fig. 8.10). The back-azimuth at the

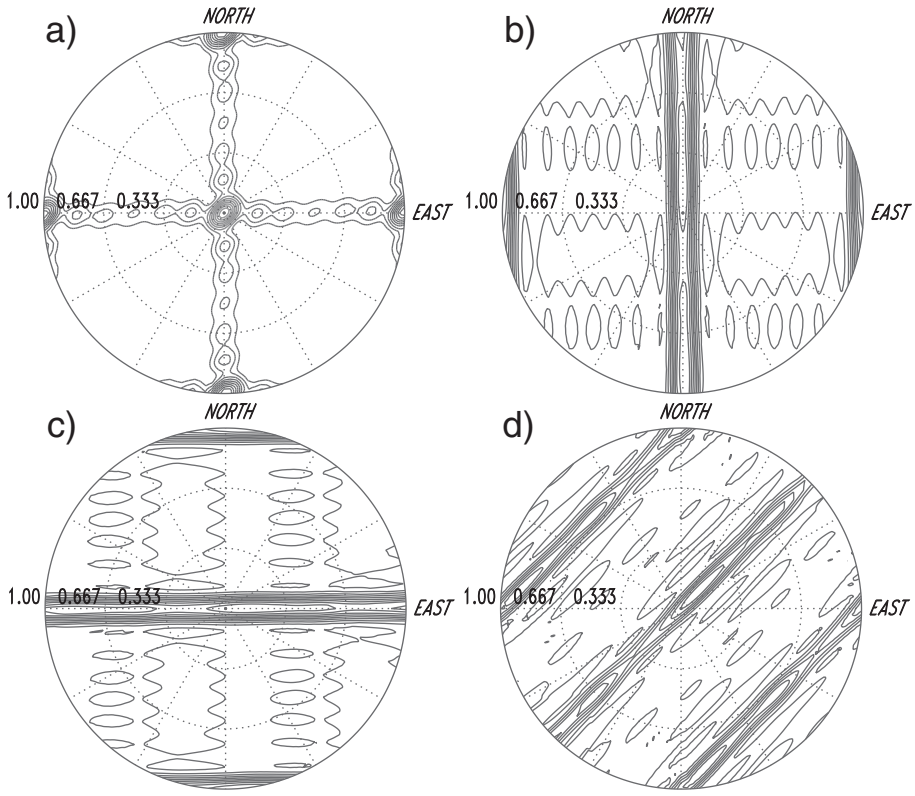


Figure 8.9 Polar plots of contoured beam power as a function of azimuth (angle) and wavenumber (radius) for a synthetic array (Fig. 8.8). The maximum radial wavenumber is 1 km^{-1} in all cases. (*a*) Zero-slowness (vertical) arrival. The cross-shaped pattern is the response function of the array; the power peak is at zero wavenumber. Aliased peaks also exist N, S, E and W bearings at $k = 1\ \mathrm{km}^{-1}$. (*b*) Arrival along a N–S bearing. The ridge extending N–S includes the origin and represents the unaliased signal. (*c*) Arrival along an E–W bearing. Again, the ridge extending E–W includes the origin and represents the unaliased signal. (*d*) Arrival along 45°. Aliasing is more evident in this case because the wavenumber is 0.707 km^{-1}. (For color version, see Plates section.)

approximate center of the network, station ESK, is 354.4°, which confirms the detection made by BBFK.

With this approach information in hand, the BEAM command may be used to form a trace aligned along that azimuth at a reference station located at 55.3167N, 3.2050W,

```
SAC> beam bearing 355 vel (1 / 0.0227) ref 55.3167 -3.2050
```

with the display shown in Figure 8.12. To examine the waveform details, use the BEAM WRITE option to write a separate trace file, read it in, and zoom in on the waveform using PPK.

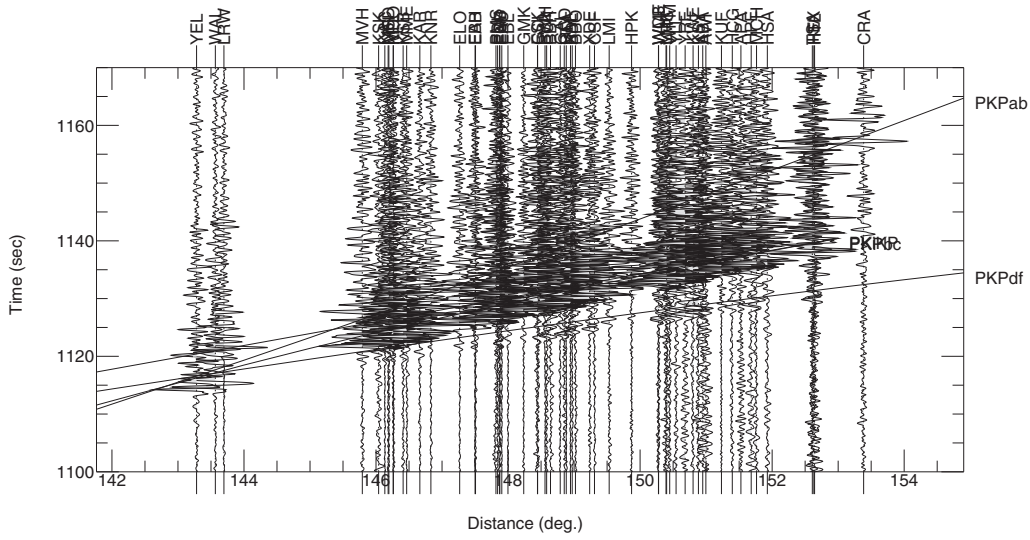


Figure 8.10 The record section shows a teleseismic arrival recorded by a regional seismic network that constitutes an array. The superimposed travel-time curves show that the arrivals PKPdf, PKPbc and PKPab are well recorded across the array. (For color version, see Plates section.)

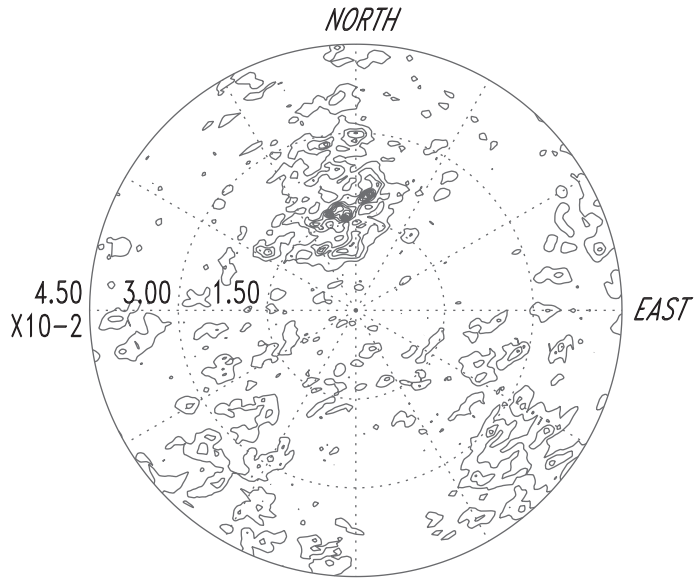


Figure 8.11 BBFK analysis of the array data (see Fig. 8.10) shows a peak along a northerly bearing and at wavenumber slightly larger than 1.5×10^{-2} km^{-1}. (For color version, see Plates section.)

8.5 TRAVEL-TIME ANALYSIS

It is not immediately apparent but SSS also has uses when applied to single station, three component data. The TRAVELTIME command is able to add theoretical arrival times to traces. The traces are usually from stations at different ranges from a single event. Thus

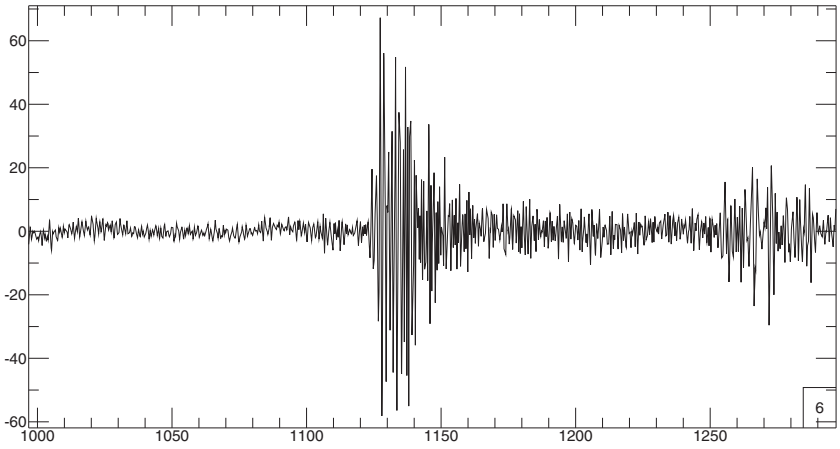


Figure 8.12 Array beam formed from BBFK analysis of the array data (see Fig. 8.10) using the spectral peak's back-azimuth and wavenumber. The entire trace is shown by the BEAM display, but it can be saved as a file and its features examined using READ followed by PPK.

the travel times document the moveout of an arrival with range. If the traces just happen to be three traces at the same station, well, so be it:

```
SAC/SSS> help traveltime

 SUMMARY:
 Computes travel-time curves for pre-defined models or reads
 travel-time curves from ascii text files.

 SYNTAX:
    TRAVELTIME [MORE] MODEL <name> [DEPTH <depth>]
        [PICKS <n>] [PHASE <phs> ...]

    TRAVELTIME [MORE] [DIR CURRENT|<dir>] CONTENT <cont>
        [HEADER <n>] [UNITS DEGREES|KM] [PICKS <n>] <file> ...

    TRAVELTIME [DIR CURRENT|<dir>] [PICKS <n>] TAUP <file>
```

The PICKS option is the key one. The various forms of the command will assign arrivals to a sequence of file header variables starting with T<n>. SAC labels the picks with the phase names. When plotted with PLOT1 as a three-component set, the predicted arrival times will be shown as labeled picks. Up to ten picks may be added to each trace in this way (to T0 ... T9).

The various forms of TRAVELTIME allow theoretical travel-time picks to be provided in different ways. Tables for the *iasp91* (Kennett and Engdahl, 1991), *ak135* (Kennett *et al.*, 1995) and *sp6* (Morelli and Dziewonski, 1993) models are built in, and naming the proper model in the first command form supplies their predicted values for the pick values for the chosen phase names. The other forms read travel times from a file, either as a list of range versus time or written by the Tau-p toolkit (Crotwell *et al.*, 1999).

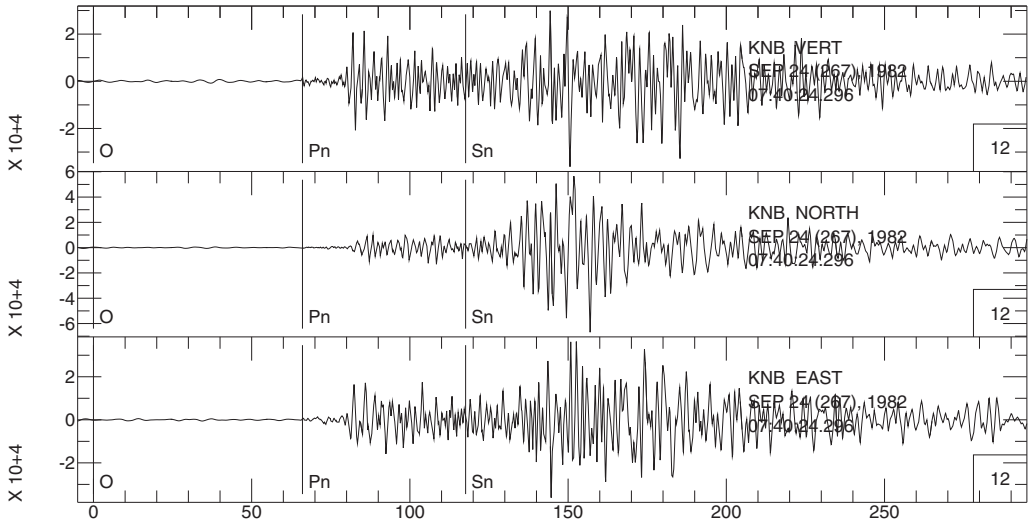


Figure 8.13 Three-component data from a single station with theoretical travel-time picks added by the TRAVELTIME command. PLOT1 adds the picks and their phase name labels to the combined trace plot.

Figure 8.13 shows a plot resulting from a set of picks arranged by the following commands:

Ex-8.11

```
SAC> datagen sub regional knb.[z,n,e]; rmean
SAC> sss
SAC/SSS> traveltime phase Pn Sn picks 0
SAC/SSS> qs
SAC> plot1
```

This is a simple way to add theoretical pick information to a plot. Another way is to use a SAC macro to make a three-component plot annotated (using PLOTC) with predicted arrivals from travel-time models. This way is more complex but does not involve using the scarce resource of file header variables for plot-related information. See Figure 11.2 for an example of this type of display.

CHAPTER
NINE

Spectral estimation in SAC

9.1 SPECTRAL ESTIMATION

Spectral estimation is the task of taking a time series and decomposing it into its component frequencies. The total length of the time series and, if it has been sampled at a fixed time interval, the inter-sample spacing control the minimum and maximum frequencies that the time series contains. Different methodologies may be used to estimate the power at each frequency at constant intervals between these bounds. This so-called power spectrum provides a way to quantitatively characterize the frequency content of the time series. The information is usually presented in the form of a graph of power versus frequency.

The power spectrum informs further processing avenues for the time series. Usually this involves discrimination of any signal in the spectrum from noise. If the signal is of high quality, the signal's power will dominate the noise. Thus the power spectrum will be peaked in a frequency band containing the signal. If the goal is to design a filtering strategy to minimize the noise, this analysis will suggest the type of filter and the corner frequency to use. Another application might be to seek tidal resonances at a coastal site based on repeated sea level measurements, or a marigram. In this case, the frequencies of the spectral peaks are the desired information, and perhaps their widths or positional uncertainties.

Spectral estimation is technically complex due to the characteristics of the signal under study. Consequently, SAC provides a suite of analysis methods applicable to different signal types (transient versus stationary) and different properties of the spectrum (its broad shape or its narrowband features). These methods, power density (PDS), maximum likelihood (MLM) (Lacoss, 1971) and maximum entropy (MEM) (Burg, 1972), represent the state of the art in about 1985. None of the authors claimed particular expertise in spectral estimation, but in the field of seismology the only later developments unacknowledged by SAC of which we are aware are multi-taper methods (Thomson, 1982), estimation of the spectra of

unequally sampled time series (Scargle, 1989), and wavelet methods for decomposing time series (Daubechies, 1992).

9.2 THE SPECTRAL ESTIMATION SUBPROCESS

SAC wraps spectral estimation into a separate subprocess (see Section 8.1). The reason is that only a small set of commands applies to spectral estimation, and those commands must be followed strictly.

The command `SPE` enters the subprocess. In it, SAC makes a spectral estimate for only one trace at a time. If the subprocess is invoked with more than one trace in memory, SAC will complain. Similarly, no trace in memory also elicits a complaint.

The spectral estimate begins with the calculation of the autocorrelation function of the trace. The autocorrelation itself may be examined or saved, but it is only an intermediate step to the spectral estimate itself. In the next step, select and apply one of the three estimation methods, PDS, MEM or MLM, to operate on the stored autocorrelation and produce the spectrum. Finally, view the spectrum by plotting it, or save it as a SAC trace in a file. Saved as files, the results from different estimation methods may be compared by using an overlay plot.

Correlation

The `COR` command calculates the autocorrelation function. SAC assumes that the trace represents a sample of a stationary statistical process. Consequently, any part of the trace is – spectrally – just like any other. The self-similarity means that the trace may be chopped into shorter segments whose spectral estimates may be averaged together to reduce the uncertainty of the whole trace's spectrum. However, this reduced uncertainty trades off with a decreased frequency resolution of the spectrum because the shorter segments have a wider frequency stride in their spectra.

The `COR` command therefore allows control over the number of segments and their length, each called a *window*. Arbitrary segmentation cuts of the trace will not have zero endpoints, which will lead to high-frequency spectral artifacts unless they are tapered. Thus `COR` offers a choice of tapers on each segment: Hamming, Hanning, cosine, triangle (Bartlett) and rectangle (or boxcar, or, equivalently, none). The `COR TYPE` option chooses the window, and the `COR LENGTH` and the `COR NUMBER` select the window length and number in the trace.

Some high-resolution estimation methods, particularly MLM, are sensitive to the high-frequency content of the data where specific frequencies might have near-zero power. This leads to apparent instability in the spectral estimate and is remedied by smoothing out the high frequencies. In the time domain, this corresponds to adding high-frequency noise to the trace and is therefore called *prewhitening*. Use the `COR PREWHITEN ON` option for prewhitening. Unfortunately, this choice actually changes the data in the trace itself, and the data must be re-read to repeat `COR` with prewhitening off.

The normalization for the autocorrelation function depends on whether the signal is stationary or transient. SAC gives a choice of normalization through the `COR TRANSIENT` or the `COR STOCHASTIC` options.

Once the autocorrelation is available, `PLOTCOR` will plot it. In itself, the autocorrelation is not very interesting. It is an oscillatory function whose maximum is at zero lag

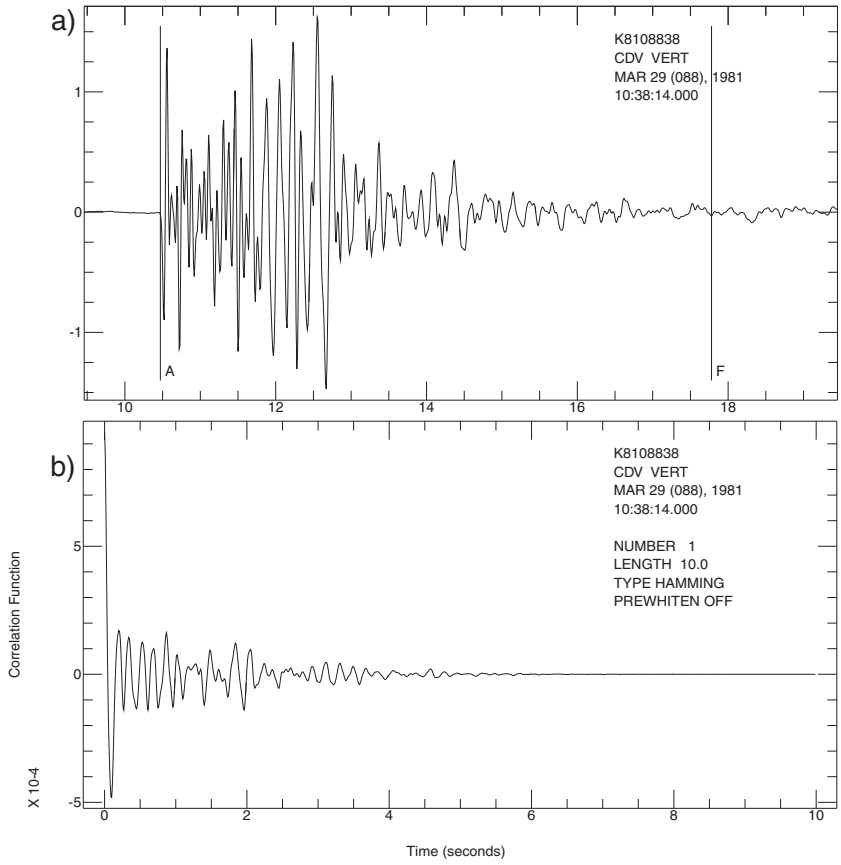


Figure 9.1 (*a*) The now-familiar seismic trace of built-in data provides a time series to show the structure of the autocorrelation function. (*b*) After COR calculates the autocorrelation, a plot by PCOR shows a peak at zero lag and an oscillatory but decreasing amplitude with increasing lag out to the window length of 10 s (the default value). Information about the COR command options appears in the label in the window.

and that decays with increasing lag. Because it is symmetric with respect to lag, positive lags contain all the autocorrelation information (Fig. 9.1). These commands will produce an autocorrelation plot of built-in data in SAC:

Ex-9.1

```
SAC> fg seismogram; rmean    ;* Data
SAC> spe                     ;* Start subprocess
SAC/SPE> cor                 ;* Default parameters - no windowing
SAC/SPE> pcor                ;* plot
```

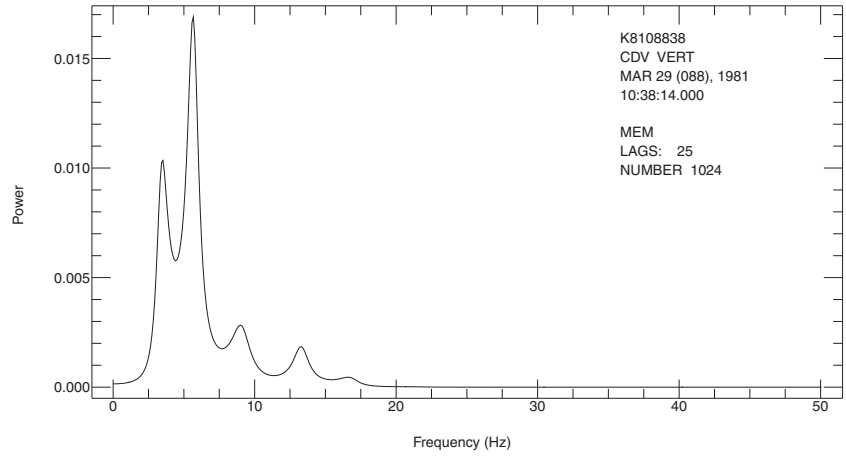


Figure 9.2 The spectrum resulting from a MEM estimate on the trace data shown in Fig. 9.1. The MEM command calculated the maximum entropy spectrum from the correlation function. PSPE plots the spectrum.

Spectrum

Once the autocorrelation is available, the spectrum follows directly from it. SAC makes the spectrum estimate by naming the method to be used: PDS, MEM or MLM. The commands recognize the NUMBER option that gives the number of frequency points in the spectrum. This should be a power of 2 that is ≥ 512. The high-resolution methods require an autoregressive model order as well, but the default is usually satisfactory. To change it, use the ORDER option. The PLOTPE command is a diagnostic to help select the order for the maximum entropy estimate. The prediction error decreases rapidly with increasing order. The plot helps to select the minimum order required to model the features in the data. Using YLOG and XLOG may help to better identify breakpoints in the curve.

PDS provides further controls on the tradeoffs inherent in spectral estimation. This method's estimated spectrum tends to behave like the true spectrum convolved with the Fourier transform of the window used on the correlation function. Thus PDS provides control over the window shape (the same available with COR TYPE) and the window length (the equivalent of the ORDER parameter for the high-resolution methods, measured in either SECONDS or LAGS).

Once the spectrum is available, the PLOTSPE command will plot it. Extending the previous example, the spectral plot shown in Figure 9.2 results from the following commands:

Ex-9.2

```
SAC> fg seismogram; rmean    ;* Data
SAC> spe                     ;* Start subprocess
SAC/SPE> cor                 ;* Default parameters - no windowing
SAC/SPE> mem                 ;* Maximum entropy spectrum
SAC/SPE> pspe                ;* plot
```

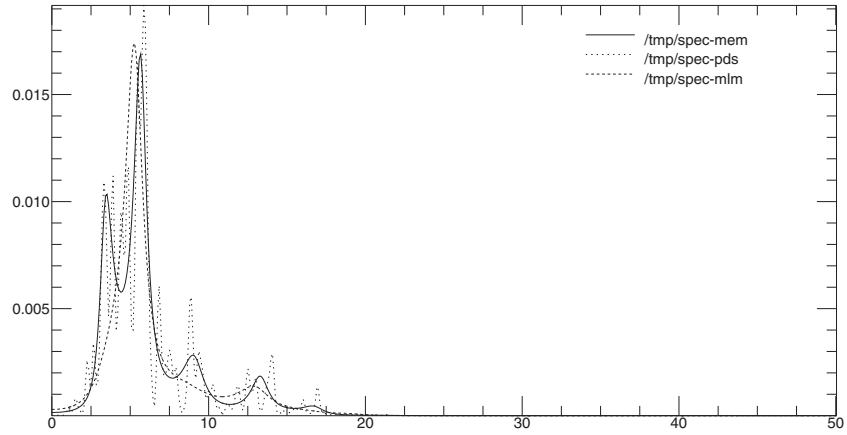


Figure 9.3 The plot shows overlaid spectra resulting from PDS, MEM and MLM estimates on the data trace shown in Figure 9.1. WRITESPE wrote out the individual estimates, which READ and PLOT2 gathered into a single plot. Different line styles denote the different methods.

Estimating the spectrum using any method does not change either the original data or the correlation function (see the caveat about prewhitening, however). Thus different methods may be tried on the same trace and viewed until the result is satisfactory.

Saving the correlation and the spectrum

Both the autocorrelation function and the spectrum may be saved as SAC files. This is principally for plotting use, because the autocorrelation is intrinsically uninteresting. However, because the autocorrelation contains all the information needed to estimate a spectrum, it is a compact way to save spectral information.

Use WRITECOR and READCOR to write and read the autocorrelation function. Autocorrelation information is saved in the file at both positive and negative lags. Thus, if read in as a SAC time series, it will be padded to a power of 2 in length and will appear to have a time-reversed copy of the positive lag autocorrelation at its end, which actually is the negative lag values.

The power spectrum estimate is the more useful quantity for an analyst. The details of these traces might be the subject of further analysis or scrutiny, possibly using PLOTPK, or to which to fit a spectral slope for an attenuation analysis procedure. The WRITESPE command writes out the spectrum. Figure 9.3 is a trivial example of what is possible with WRITESPE. The following commands estimate the spectrum of our favorite time series using each of the three methods provided by SAC. The plot shows a comparison of the estimates overlaid in a single plot using PLOT2 after WRITESPE.

Ex-9.3

```
SAC> fg seismogram; rmean          ;* Data
SAC> spe                           ;* Start subprocess
SAC/SPE> cor                       ;* Default parameters
```

```
SAC/SPE> mem                      ;* Maximum entropy spectrum
SAC/SPE> writespe /tmp/spec-mem   ;* save
SAC/SPE> pds                      ;* Power density spec
SAC/SPE> writespe /tmp/spec-pds   ;* save
SAC/SPE> cor prewhiten on         ;* Stabilize MLM
SAC/SPE> mlm                      ;* Maximum likelihood spec
SAC/SPE> writespe /tmp/spec-mlm   ;* save
SAC/SPE> qs                       ;* Leave subprocess
SAC> read /tmp/spec-[mem,pds,mlm];* re-read spectra
SAC> line list 1 2 3 increment on;* ID by line type
SAC> plot2                        ;* overlay
SAC> line previous                ;* restore line options
```

CHAPTER
TEN

Three-dimensional data in SAC

10.1 THE CONCEPT OF 3D DATA

SAC has facilities for handling a basic type of three-dimensional (3D) data, which are function values evaluated on a regular grid of (X,Y) positions. This type of data can be viewed as evenly sampled in space and fits naturally into SAC file types. The file consists of the string of samples at each grid point, and file header information that flags it as 3D gives the X and Y dimensions of the grid. SAC calls data of this type `XYZ`.

The simplest visualization of 3D data is map elevation data. Imagine a section of land whose elevation is specified on a grid every 100 m eastwards and 100 m northwards. Over a plot of land 1 km by 1 km, the elevation could be described by a string of 121 values, the first 11 values along the westernmost traverse and the final 11 along the easternmost traverse.

Another type of data that can be represented in 3D is a spectrum as a function of time. If equal lengths of time are cut from a seismogram, the number of points in their FFTs will be the same. If the interval between spectral samples is constant, these values will also form a grid.

A final type of data inherently 3D is a misfit function evaluated over a two-dimensional (2D) grid of points. The misfit minimum within the sampled parameter space yields the optimum combination of parameters that best fit some observations, and the shape of the minimum provides a measure of the joint uncertainty of the parameter estimates.

10.2 SPECTROGRAMS

A spectrogram is a depiction of the time variation of a signal spectrum. In a time window of fixed length, the spectrum of that subset of the time series will have a fixed number

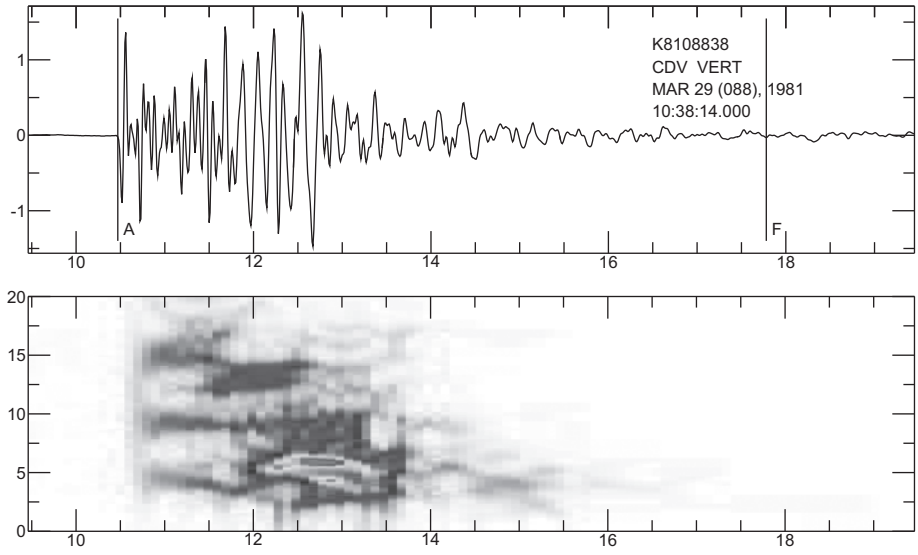


Figure 10.1 Spectrogram of a seismogram: (top) the time series and (bottom) the spectrogram. The X axis is time as it runs through the seismogram. The Y axis is frequency (Hz). In a time window of 2 s recalculated every 0.1 s through the seismogram, the color represents the spectral power at each frequency. (For color version, see Plates section.)

of frequencies. As the time window marches through the time series, the evolution of the spectral content provides a visual assessment of a signal onset and its power at each frequency. This visualization provides a way to graphically assess the dominant frequency of a body wave arrival or the dispersion curve of a surface wave arrival.

SAC provides the SPECTROGRAM command for this use. The command's options control the length of the time window (thus its frequency content), its stride (repetition in time), and the method of spectral estimation (standard, maximum entropy or maximum likelihood). The result is a 3D SAC file that is displayed with color encoding the amplitude at each frequency (Y axis) at each point in time (X axis). The file may be saved using the SPECTROGRAM XYZ option, or, as is commonly done, displayed immediately. Figure 10.1 shows a spectrogram produced using the commands below:

Ex-10.1

```
SAC> fg seismogram; rmean           ;* Data
SAC> cuterr fillz; cutim (&1,b - 1) (&1,e + 1)   ;* Pad
SAC> bf                             ;* Multi-plot display
SAC> yvport 0.55 0.95
SAC> xlim (&1,b + 1) (&1,e - 1); p1; xlim previous ;* Data plot
SAC> yvport previous
SAC> yvport 0.1 0.48
SAC> spectrogram slice 0.1 ymax 20 method mlm      ;* Spectrogram
SAC> yvport previous
SAC> ef                             ;* end display
```

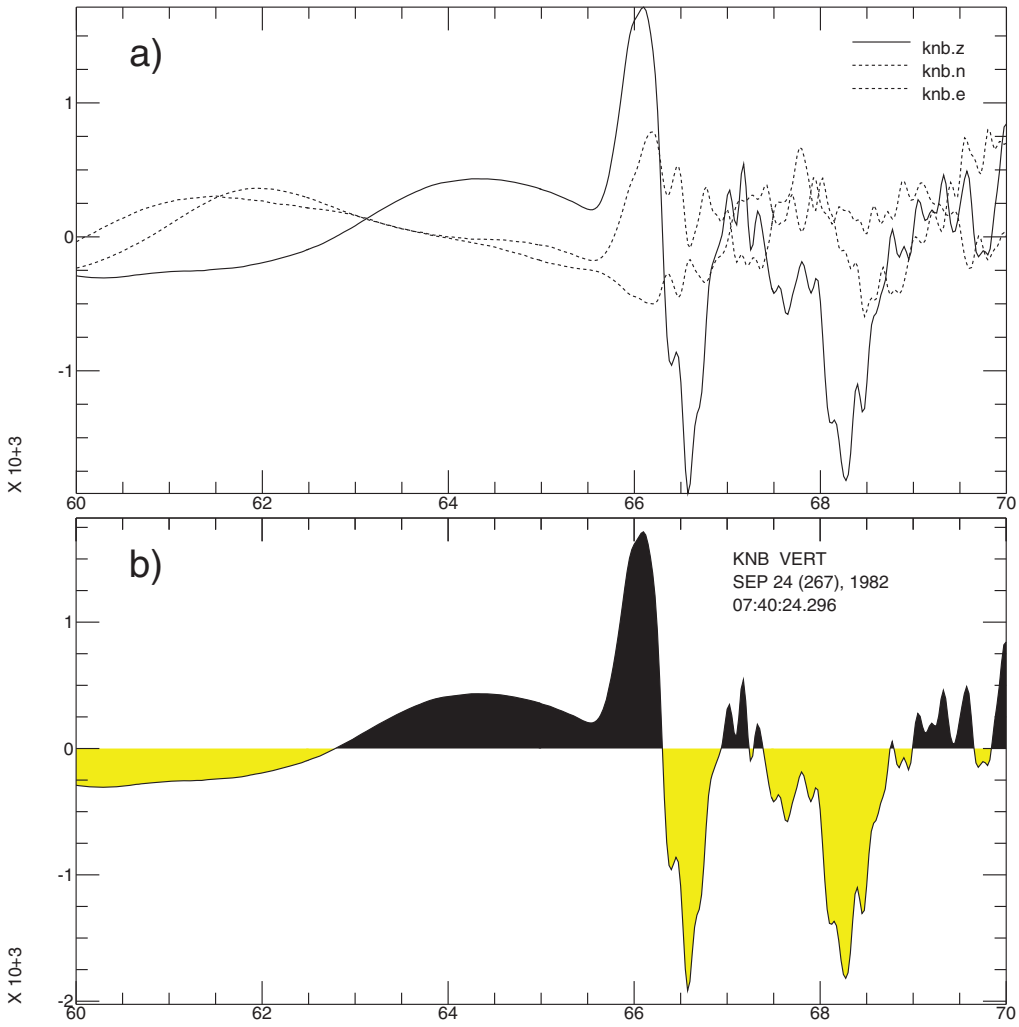


Figure 4.2 The three seismograms plotted by PLOT2 (*a*) show how different line styles enhance comprehension of the display. The line style for the horizontal component traces is different from the vertical component trace. PLOT2 shows this in the legend at top right in the display frame. Another line style feature that SAC provides to improve trace feature visibility is color-filled traces (*b*). The positive and negative areas may be filled with the same color, different colors, or limned selectively.

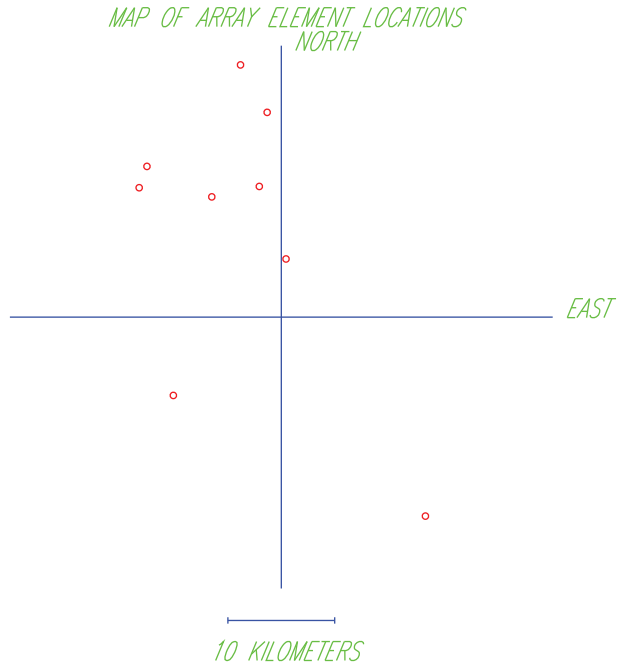


Figure 8.7 This plot shows an X–Y map of stations in a local network recording an earthquake, produced by `ARRAYMAP`. The origin of the map is the epicenter, and circles show individual station locations.

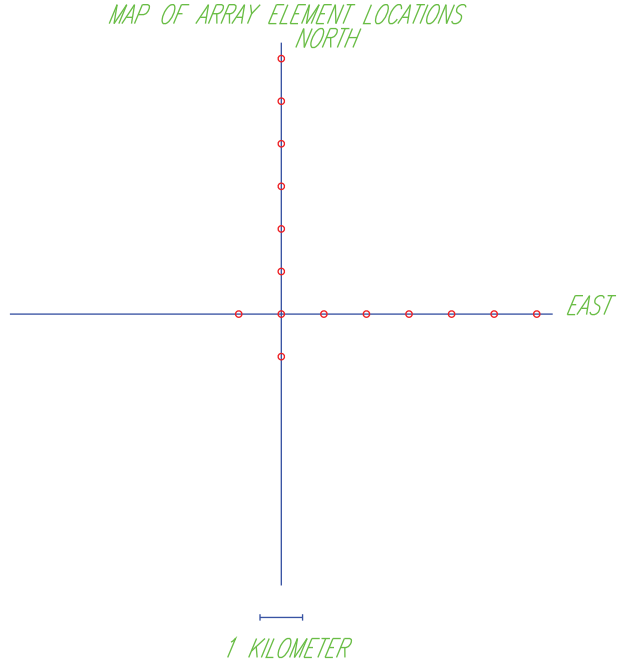


Figure 8.8 This plot shows an X–Y map of the example local array whose synthetic response is explored by the `ex-bbfk.m` macro (see text). The plot is produced by `ARRAYMAP`. Elements are spaced every kilometer along two asymmetric arms, similar to the UK Atomic Energy Agency design.

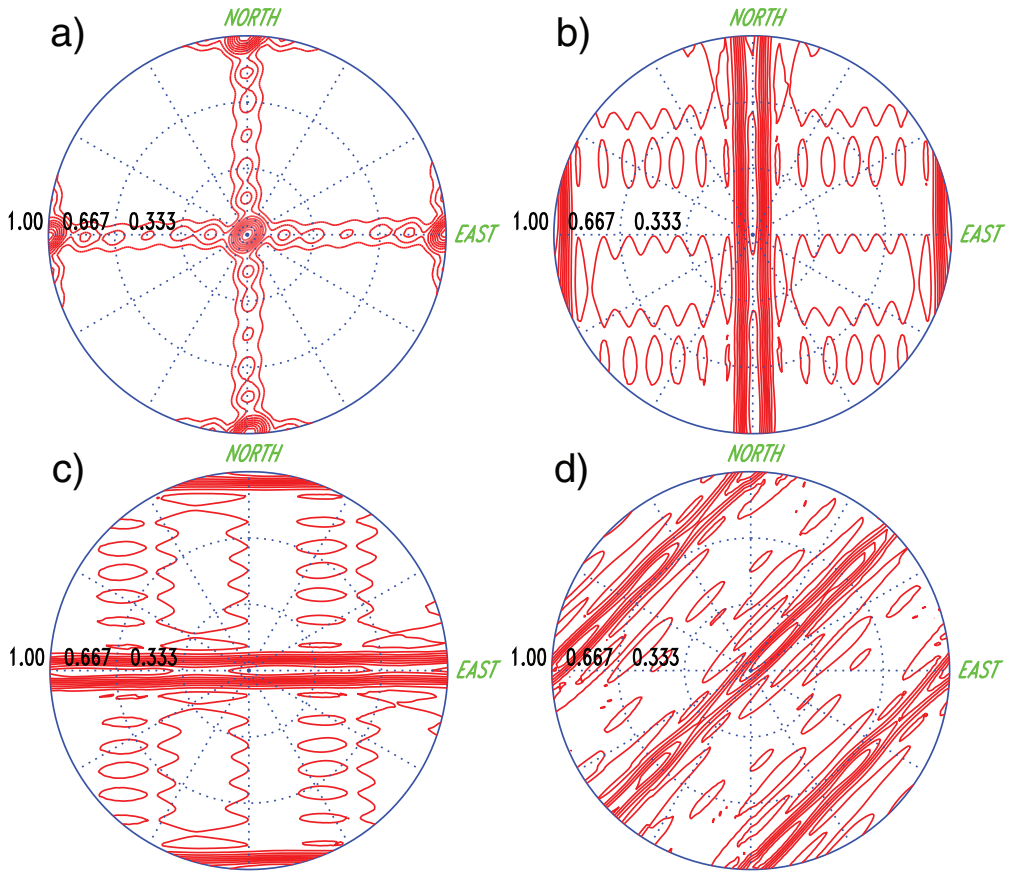


Figure 8.9 Polar plots of contoured beam power as a function of azimuth (angle) and wavenumber (radius) for a synthetic array (Fig. 8.8). The maximum radial wavenumber is 1 km^{-1} in all cases. (*a*) Zero-slowness (vertical) arrival. The cross-shaped pattern is the response function of the array; the power peak is at zero wavenumber. Aliased peaks also exist N, S, E and W bearings at $k = 1$ km^{-1}. (*b*) Arrival along a N–S bearing. The ridge extending N–S includes the origin and represents the unaliased signal. (*c*) Arrival along an E–W bearing. Again, the ridge extending E–W includes the origin and represents the unaliased signal. (*d*) Arrival along 45°. Aliasing is move evident in this case because the wavenumber is 0.707 km^{-1}.

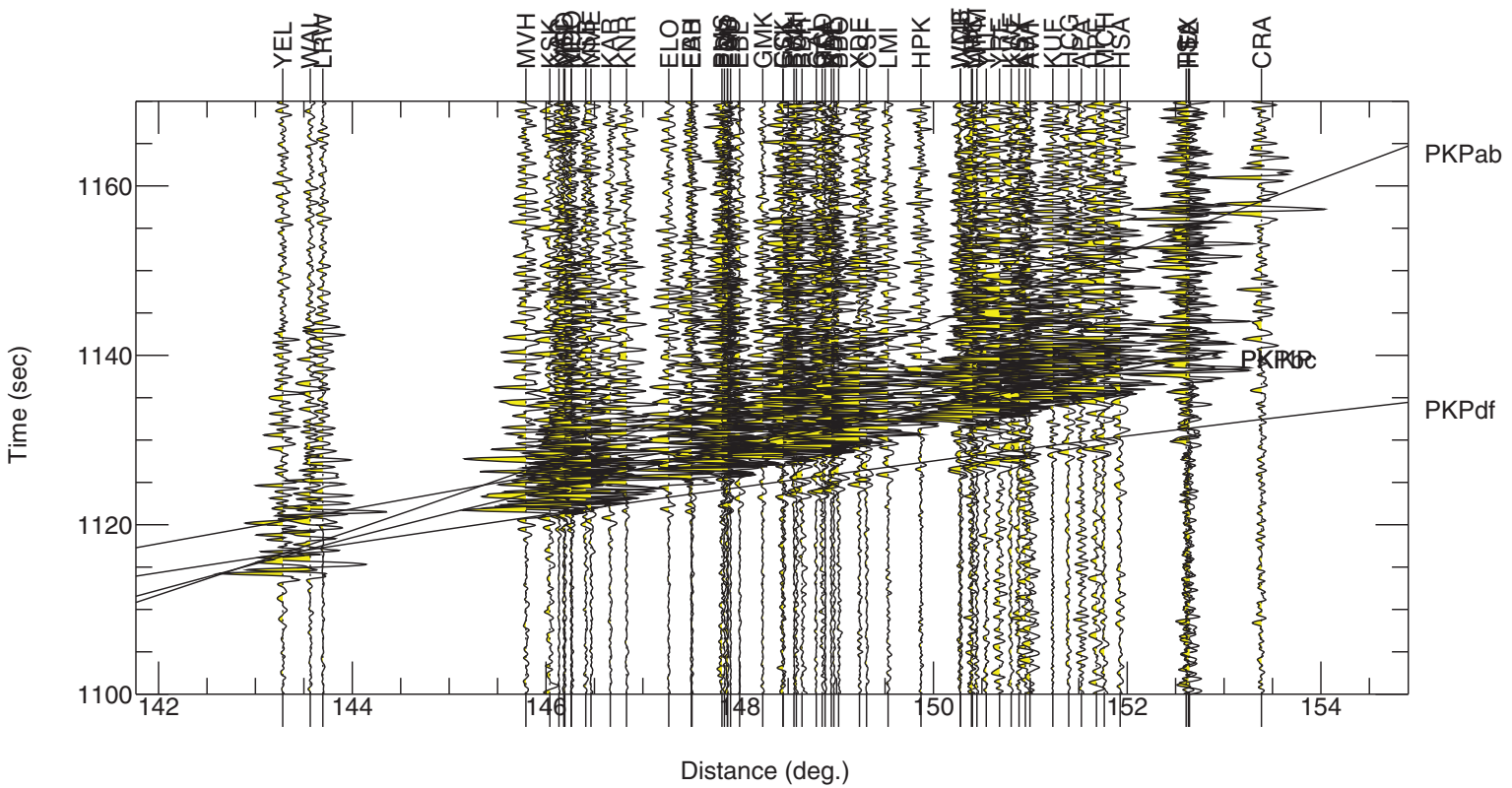


Figure 8.10 The record section shows a teleseismic arrival recorded by a regional seismic network that constitutes an array. The superimposed travel-time curves show that the arrivals PKPdf, PKPbc and PKPab are well recorded across the array.

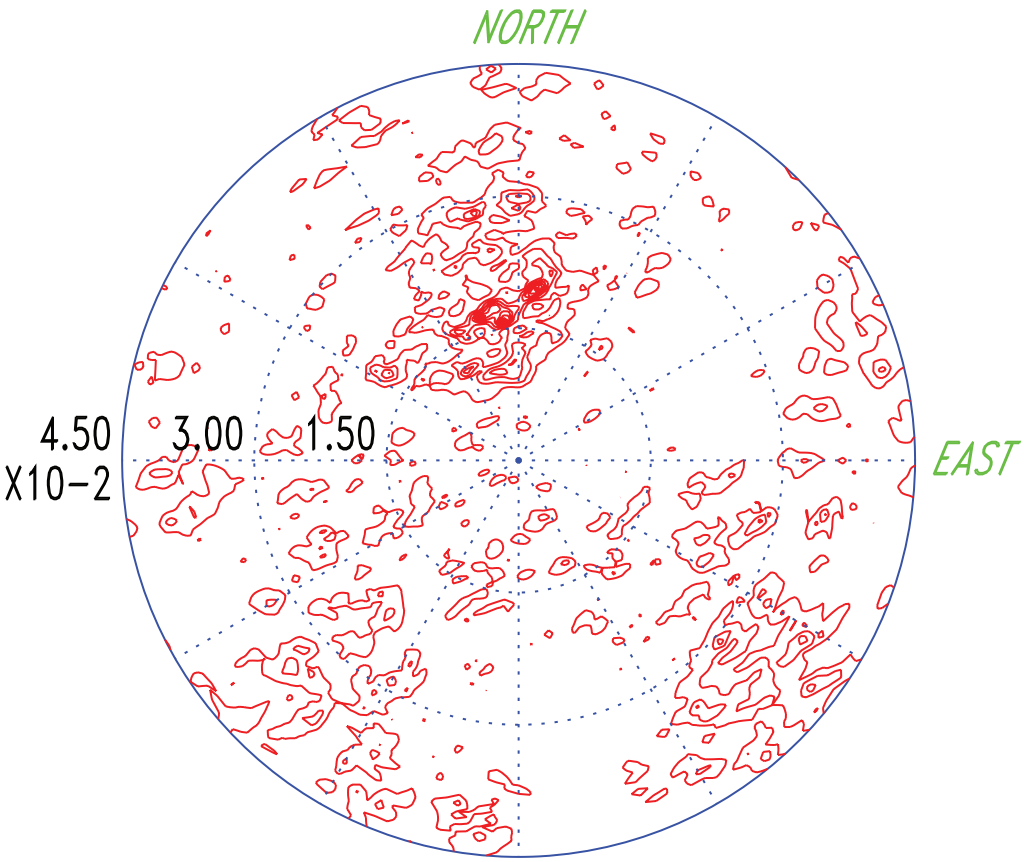


Figure 8.11 BBFK analysis of the array data (see Fig. 8.10) shows a peak along a northerly bearing and at wavenumber slightly larger than 1.5×10^{-2} km^{-1}.

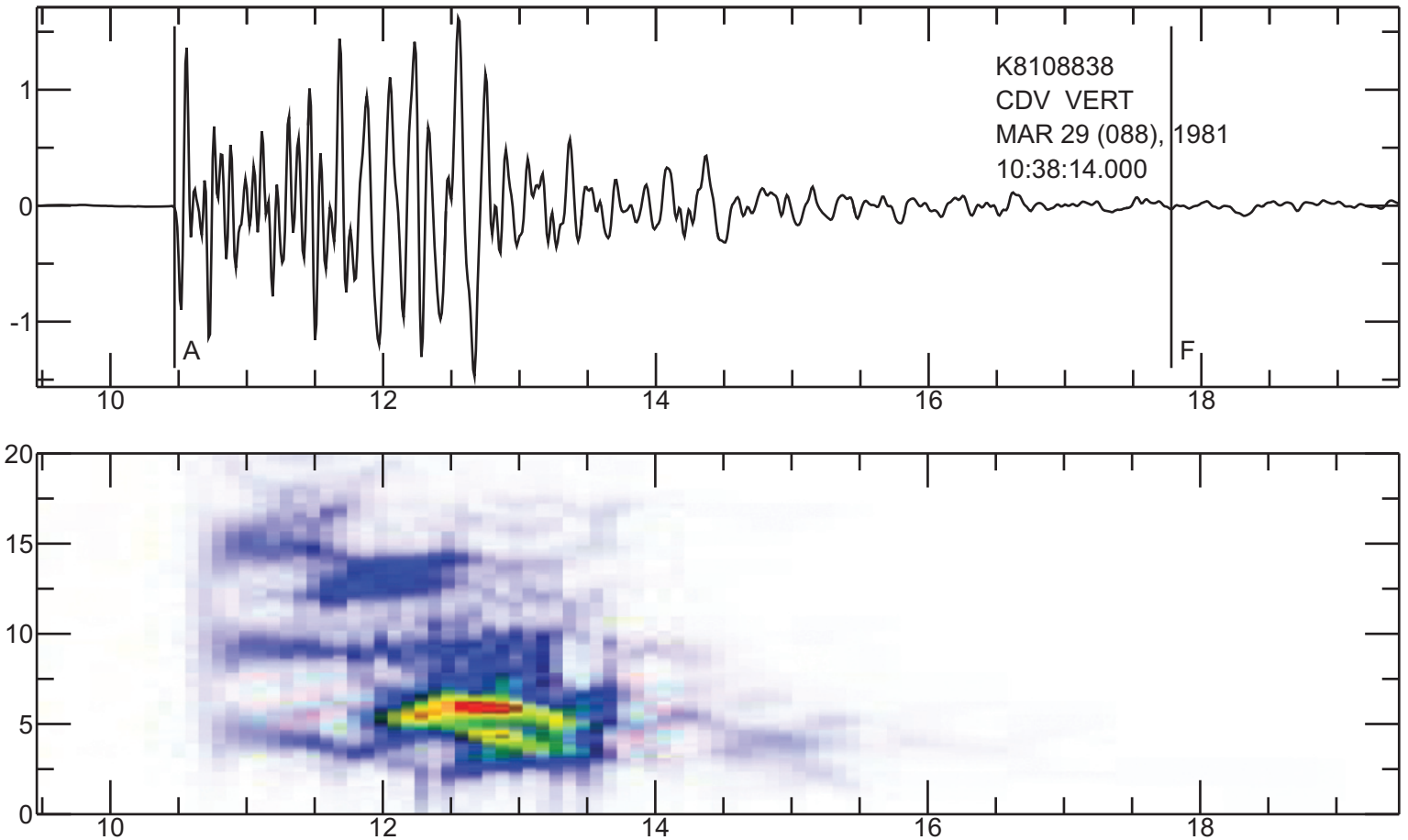


Figure 10.1 Spectrogram of a seismogram: (top) the time series and (bottom) the spectrogram. The X axis is time as it runs through the seismogram. The Y axis is frequency (Hz). In a time window of 2 s recalculated every 0.1 s through the seismogram, the color represents the spectral power at each frequency.

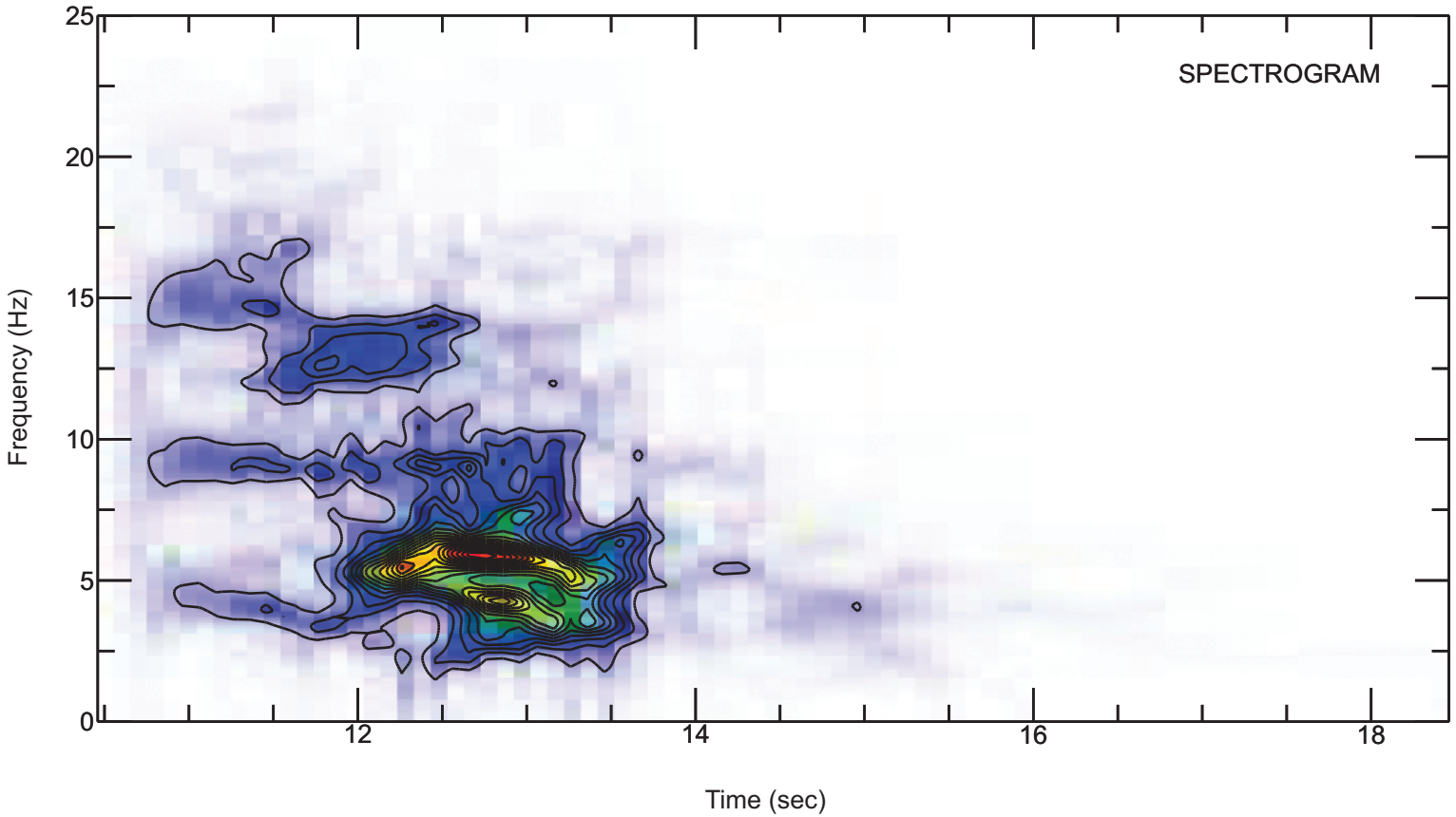


Figure 10.4 XYZ file data may be usefully displayed in alternative forms by SAC. The figure shows a spectrogram, saved as a 3D file, plotted in color with the `GRAYSCALE` command. It is overlaid with contours of spectral power. This provides both a quantitative report on the spectrogram and a qualitative graphical summary.

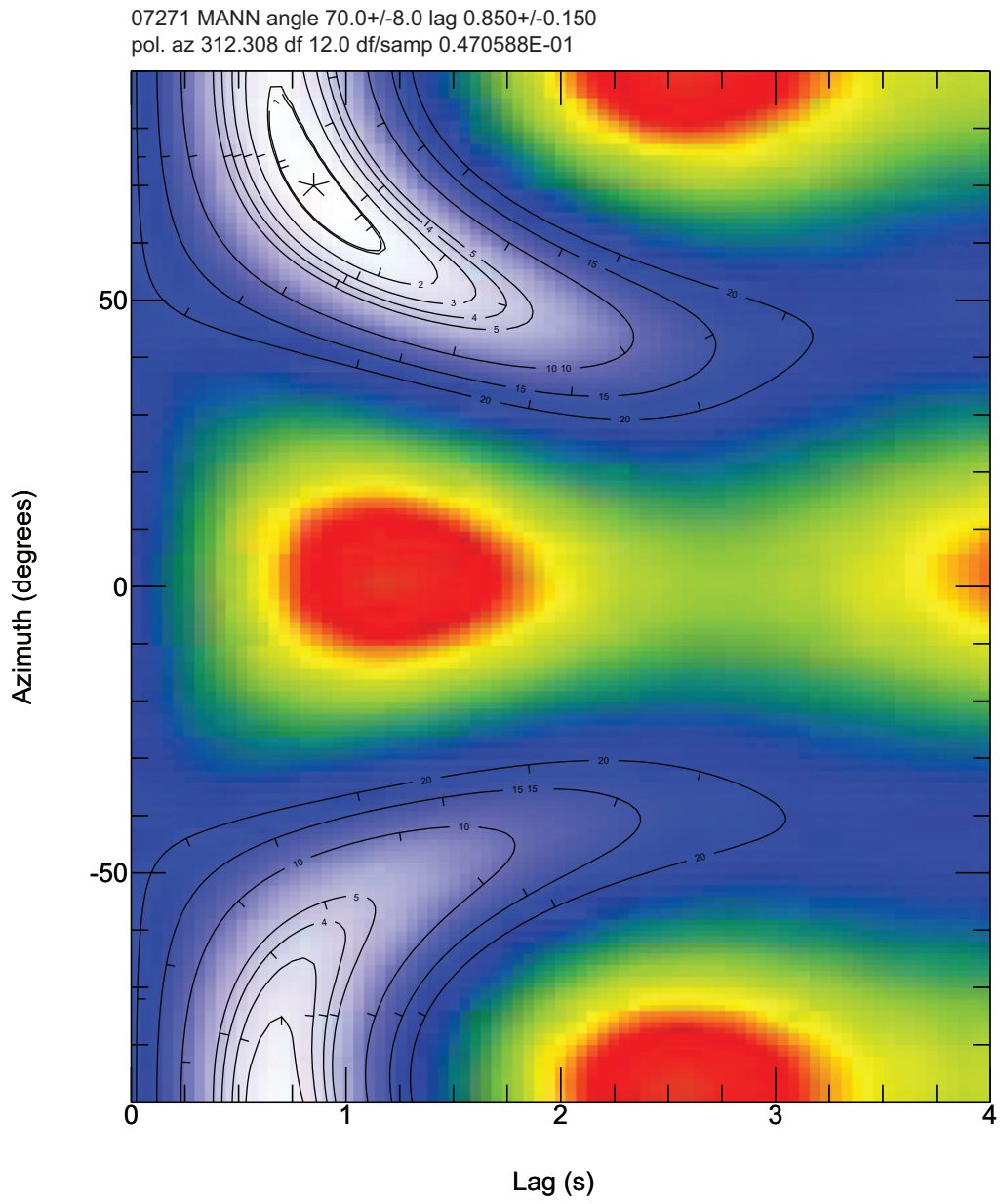


Figure 11.4 Results of the grid search over δt and ϕ. The optimum splitting parameters are shown by the star (top left). The 95% confidence region is shown (double contour), along with multiples of that contour level.

This spectrogram was plotted directly. However, the SPECTROGRAM XYZ option will replace the data in memory with an XYZ-type file that may be written or processed using further SAC commands.

10.3 CONTOUR PLOTS

Any file of 3D data of type XYZ can be treated as elevations over a 2D surface. A natural way to visualize the data is by contouring it, and SAC provides the command CONTOUR to do this. A basic contour plot of SAC's built-in contour data is simple to make

```
SAC> datagen sub xyz volcano.xyz
SAC> contour
```

and results in the plot shown in Figure 10.2. This is a basic plot without any ornamentation on the contours, or even an indication of which contour is high and which is low. SAC provides a plethora of commands to improve contour plot annotation.

In addition to adding axis labels (XLABEL, YLABEL) and a title (TITLE) to a plot, SAC provides ways to affect the contour lines themselves. ZLEVELS sets the values at which to draw the contours. It is possible to set a low and high range for plotting, with either a number of lines in that range or an increment between levels. Alternatively, list the desired levels specifically. After defining a set of levels, associate a drawing color (see ZCOLORS) and a line style (see ZLINES) with each. To attach a label to each level, use ZLABELS. It provides the text or format for each level's label and its size and inter-label spacing rules. (Good labeling on contour plots is an art that requires much experimentation to achieve a pleasing result. The features of ZLABELS will reward close study.) To show the gradient sense, SAC provides a ZTICKS to place ticks perpendicular to the contour in either the uphill or downhill direction.

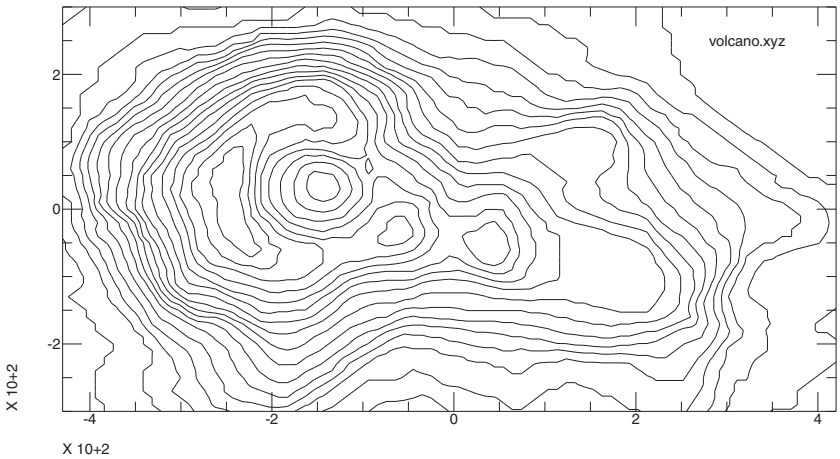


Figure 10.2 Contour plot of a topograpic feature. The basic contour plot is not labeled, and the contour levels are solid lines unlabeled with the contour value. This provides a quick overview of the data, but is not quantitative.

Two of SAC's built-in contouring demonstrations illustrate some of these features; use `ECHO ON` to see details. The command

```
SAC> macro demo contour simple
```

draws the plot shown in Figure 10.3a where `ZLINES` is used to make contour lines solid every 20 m of elevation. The 5 m contour increment (`ZLEVELS INCREMENT 5`) means that

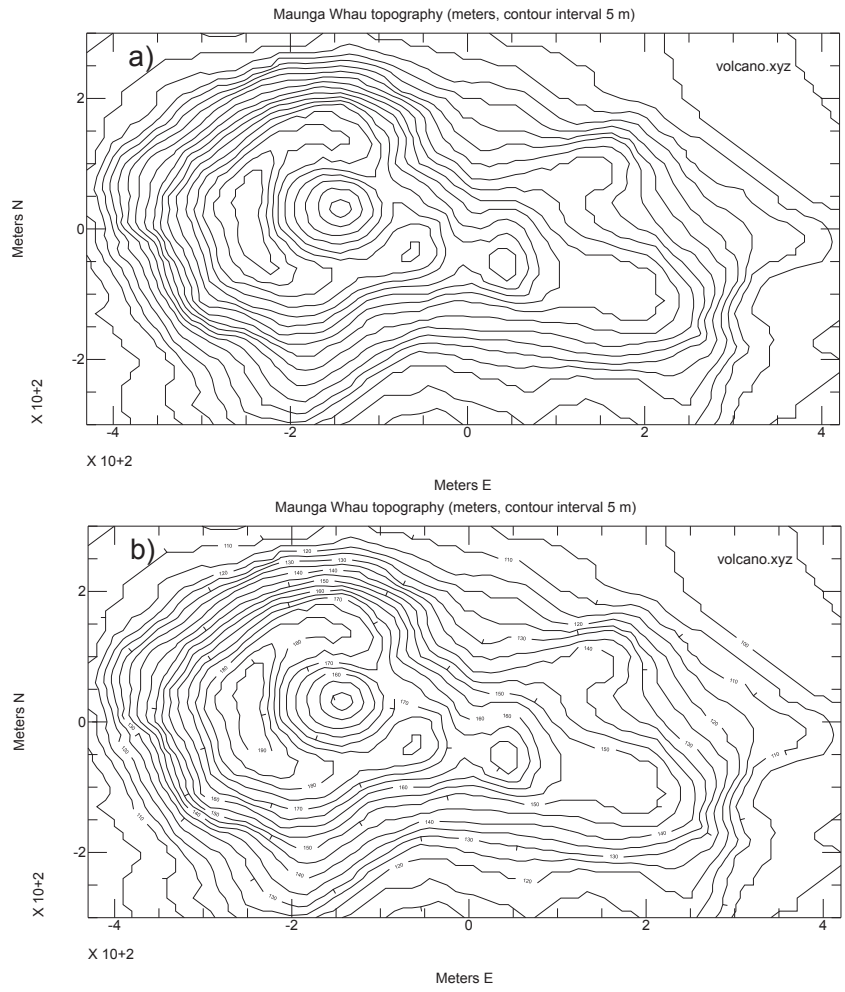


Figure 10.3 This is the same topographic feature seen previously. (*a*) The feature with different line styles that repeat every 20 m. This provides the viewer a visual estimate of the vertical scale of the feature without labeling each contour line. (*b*) The feature with the same line style for all contours. Tick marks on the downhill side of the contours clearly show the pit crater at the summit of the feature and the summit-girdling ridge of the crater. Every other contour line bears a level label.

every fourth line will lie at a multiple of 20 m. Thus, ZLINES LIST 1 2 3 4 returns to a solid line (line style 1) after cycling through styles 2–4.

The command

```
SAC> macro demo contour complex
```

uses more features of CONTOUR. Figure 10.3b shows the result. Here, the same data is displayed with tick marks showing the downhill direction on every fourth level, and every other contour level is labeled with an integer elevation.

10.4 COMPOSITE 3D DATA PLOTS

Once written into an XYZ file, 3D data can be usefully represented in composite forms. The GRAYSCALE command draws a color or grayscale image of an XYZ file using the prevailing color palette to translate Z values into color. A default palette is supplied. To set GRAYSCALE's palette to a custom palette, use the COLOR RASTER LIST command. REPORT COLOR will show the pre-defined color palettes to choose from, or a particular one may be built up from a user's own color list.

A color palette is a file containing lines in the form

```
0.667 0.000 0.000 BRICK-RED
1.000 0.000 0.000 RED
1.000 0.333 0.000 ORANGE
1.000 0.667 0.000 PUMPKIN
1.000 1.000 1.000 WHITE
0.353 1.000 0.118 ACID-GREEN
0.000 0.941 0.431 SEA-GREEN
0.000 0.314 1.000 ROYAL-BLUE
0.000 0.000 0.804 DEEP-BLUE
```

Each line represents a color. The three numbers are a red, green and blue intensity (1 = full, 0 = none) describing the color. The text following the numbers is optional, but provides a way to refer to the color by name in the LINE and COLOR command. The range of Z values encountered in a plot is mapped linearly onto the colors. Linear interpolation of the (red, green, blue) values between bracketing colors provides intermediate color values. The palette here, with white the central color, would be suitable for plotting a zero-mean value.

Color is a qualitative indicator of level because humans respond to it individually. Not only do forms of color blindedness thwart discrimination between some color palette choices, but it is hard to judge the difference between, say, the color indigo and the color violet marking a Z value boundary in a 3D plot. For quantitative use, SAC therefore provides the ability to combine contour plots with color images (or black and white images). The contour lines provide the quantitative levels, and the color provides the visual summary.

A good example of the idea is the composite spectrogram and contour plot shown in Figure 10.4, resulting from the following commands:

Ex-10.2

```
SAC> fg seismogram; rmean                  ;* Data
SAC> spectrogram slice 0.1 method mlm xyz ;* Make XYZ spectrogram
SAC> bf                                    ;* Start composite plot
```

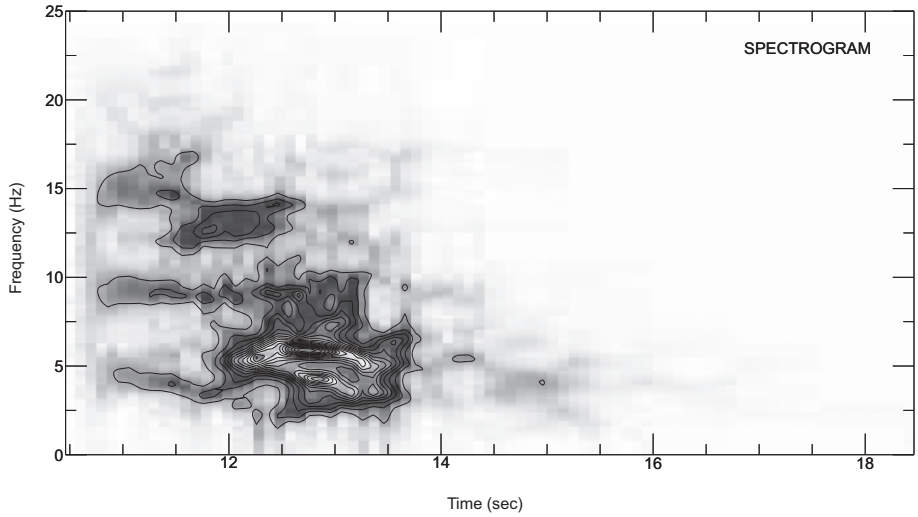


Figure 10.4 XYZ file data may be usefully displayed in alternative forms by SAC. The figure shows a spectrogram, saved as a 3D file, plotted in color with the GRAYSCALE command. It is overlaid with contours of spectral power. This provides both a quantitative report on the spectrogram and a qualitative graphical summary. (For color version, see Plates section.)

```
SAC> ylim 0 25                                  ;* Limits and labels
SAC> ylabel 'Frequency @(Hz@)'; xlabel 'Time @(sec@)'
SAC> grayscale                                  ;* Color spectrogram
SAC> ylabel off; xlabel off                     ;* Only label once
SAC> contour                                    ;* Overlay contours
SAC> ef                                         ;* Finish composite plot
```

The commands calculate a spectrogram of some of SAC's built-in data and save it as an XYZ file. The first plot (GRAYSCALE) is a color realization of the spectrogram inside a BEGIN-FRAME ... ENDFRAME pair. The second CONTOUR plot overlays the contour levels. A useful addition to this basic plot would be labels on the contour lines.

10.5 PROPERTIES OF 3D DATA

Here we turn to the technical details of 3D data. These are unimportant except for writing programs to create or handle 3D data for SAC's use. The details are mercifully brief.

3D data are Z values of function sampled on a regular X–Y grid. There is a compelling analogy between time series and this structure because a time series is also a function that is sampled on a regular time grid. Thus the only values to be stored for either are the sample values from a starting value (time or position) and an interval (time or space).

For an XYZ file, the starting value is the (X,Y) point of the grid specified by the file header variables XMINIMUM and YMINIMUM. The file header variables NXSIZE and NYSIZE give the number of grid elements along the two space axes. The grid point separations

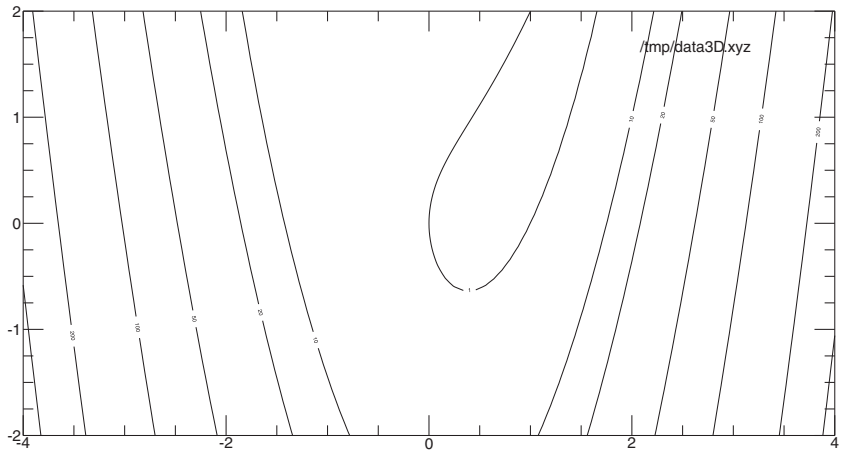


Figure 10.5 Contour plot of an XYZ data file written by the sample program "plot3d" listed in the text. The commands to produce this plot are

```
SAC> read /tmp/data3D.xyz
SAC> zlevels list 0 1 10 20 50 100 200 300
SAC> zlabels on list 0 1 10 20 50 100 200 300
SAC> contour
```

are implicit, defined by XMAXIMUM and YMAXIMUM. Thus the spatial scale is defined by (XMAXIMUM-XMINIMUM)/(NXSIZE-1) rather than as an explicit value like DELTA for a time series. The file type IFTYPE must also be set to IXYZ for these values to be recognized. (Otherwise, SAC reckons the file is a kind of time series. In fact, PLOT will plot an XYZ file – an occasionally useful feature to view the data range.) Finally, the number of points in the file, NPTS, must be set to NXSIZE×NYSIZE.

In the file itself, the data is stored along streaks of Y values with increasing X (see Section 2.4). This is congruent to the way that time series values (Y) are stored with increasing time (X).

10.6 WRITING 3D DATA FILES

Writing 3D data files is simple: first set the file header variables and then call WSAC0. The following Fortran program shows the necessary steps. First, create the data. Then call NEWHDR to create a skeleton file header. Populate the header with values using SETxHV and then use WSAC0. Figure 10.5 shows the resulting figure.

Ex-10.3

```
      program plot3D
C     Grid extends symmetrically from zero
C        for nxgr cells in X and nygr in Y
      parameter (nxgr=50,nygr=50)
      parameter (nx=1+2*nxgr, ny=1+2*nygr)
C     Parameters: depth of valley (fscl), X and Y ranges (4x2)
```

```
      parameter (fscl=1, xscl=4, yscl=2)
C     Data for grid.  Grid layout is
C        (*, 1) - lowest streak of y values
C        (*,ny) - highest streak of y values
C        (1, *) - lowest x value at any y position
C        (nx,*) - highest x value at any y position
      real data(nx,ny)

C     Create data:  Curved valley defined by a Rosenbrock fcn
      f(x,y) = (1-x)**2 + fscl*(y-x**2)**2

C     Run over grid and insert elevation information into it.

      do j=1,ny
         y = yscl*float(j-1-nygr)/nygr
         do i=1,nx
            x = xscl*float(i-1-nxgr)/nxgr
            data(i,j) = f(x,y)
         enddo
      enddo

C     Create default SAC file header; modify to make it 3-D data
      call newhdr

C     File type to XYZ for 3D data
      call setihv('IFTYPE', 'IXYZ', nerr)

C     Grid dimensions: NXSIZE, NYSIZE
      call setnhv('NXSIZE', nx, nerr)
      call setnhv('NYSIZE', ny, nerr)

C     Grid scale:  XMINIMUM, XMAXIMUM, YMINIMUM, YMAXIMUM
      call setfhv('XMINIMUM', -xscl, nerr)
      call setfhv('XMAXIMUM',  xscl, nerr)
      call setfhv('YMINIMUM', -yscl, nerr)
      call setfhv('YMAXIMUM',  yscl, nerr)

C     Number of data points, begin, delta (required but irrelevant)
      call setfhv('B', 0.0, nerr)
      call setfhv('DELTA', 1.0, nerr)
      call setnhv('NPTS', nx*ny, nerr)

C     Write data to file using present header values
      call wsac0('/tmp/data3D.xyz',data,data,nerr)
      end
```

CHAPTER

ELEVEN

Implementation of common processing methodologies using SAC

11.1 SEISMIC ANISOTROPY AND SHEAR WAVE SPLITTING

Overview

The observation of two independent, orthogonally polarized shear waves, one traveling faster than the other, is arguably the most unambiguous indicator of wave propagation through an anisotropic medium. The splitting can be quantified by the time delay (δt) between the two shear waves and the orientation (ϕ) of the fast shear wave (Fig. 11.1).

In this chapter, we review briefly the theory behind shear wave splitting, with particular focus on the popular method of Silver and Chan (1991), for which we provide source code and documentation.

11.2 SHEAR WAVE SPLITTING ANALYSIS

Seismic anisotropy can be studied via shear wave splitting throughout the Earth, ranging in depth from hydrocarbon reservoirs in the shallow crust (e.g., Verdon *et al.*, 2009) to the core and deep mantle (e.g., Wookey and Helffrich, 2008). By selecting earthquakes at angular distances $\geq 88^\circ$ from a recording site, phases such as *SKS*, *PKS* and *SKKS* can be readily isolated for analysis of upper mantle anisotropy (for a review, see Savage, 1999). We will focus on analysis of these core phases here.

Patterns of seismic anisotropy can develop due to the preferential alignment of minerals in the crust and/or mantle, the preferential alignment of fluid or melt, layering of isotropic materials, or some combination thereof (e.g., Blackman and Kendall, 1997). Several

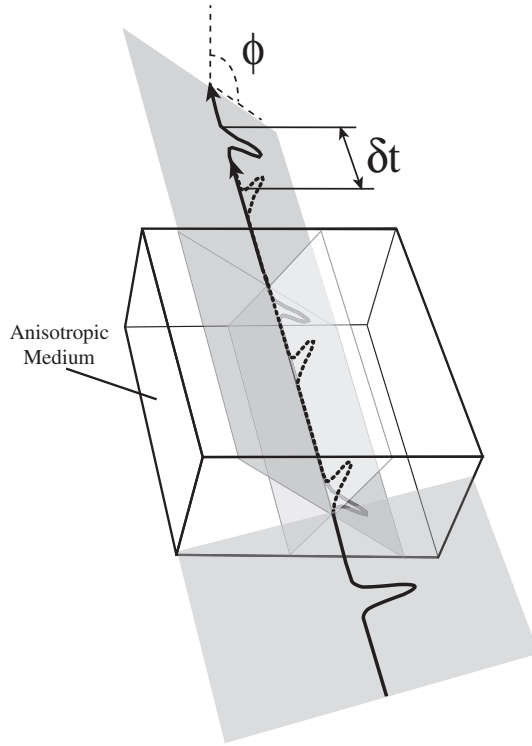


Figure 11.1 When a horizontally polarized shear wave enters an anisotropic medium, it splits into two orthogonally polarized waves. When the two pulses reach a seismometer, one can measure the polarization direction (ϕ) of the fast shear wave and the delay time (δt) between it and the later arriving "slow" shear wave. These splitting parameters can subsequently be used to characterize the anisotropic medium.

processes can develop such anisotropy, including (1) asthenospheric flow parallel to absolute plate motion (e.g., Bokelmann and Silver, 2002), (2) mantle flow around cratonic roots (e.g., Fouch *et al.*, 2000) or subducting slabs (e.g., Di Leo *et al.*, 2012), and (3) pre-existing fossil anisotropy frozen in the lithosphere (e.g., Bastow *et al.*, 2007).

For any S arrival passing through the core, the P-to-S conversion developed at the core–mantle boundary should, in an isotropic Earth, display energy only on the radial component seismogram; its particle motion should thus be linear. In the presence of seismic anisotropy, however, this assumption breaks down and energy can be found on the tangential component seismograms as well. This results in elliptical particle motion.

Parameter estimation methodologies

In principle, the goal of any shear wave splitting analysis methodology is to minimize tangential component energy and to linearize particle motion. This can be achieved in a number of ways (see e.g., Wuestefeld and Bokelmann, 2007) but by far the most common is the method of Silver and Chan (1991). This methodology rotates and time shifts the horizontal components to minimize the second eigenvalue of the covariance matrix for particle motion in a time window around the shear wave arrival. This corresponds to linearizing the particle motion and usually reduces the tangential component energy (assuming the incoming *SKS* wave is radially polarized before entering the anisotropic medium). The data is typically filtered prior to analysis with a zero-phase two-pole Butterworth filter with corner frequencies 0.04–0.3 Hz using SAC's `BANDPASS` command or with an equivalent filtering application.

Macro and auxiliary program design

The implementation of the Silver and Chan (1991) methodology is a program written in Fortran. However, interfacing with the software is required at three principal stages of the analysis:

1. visually inspecting seismic waveforms and picking analysis windows;
2. transferring data to the analysis program;
3. displaying analysis results in a graphical format.

SAC is used in each of these steps. The macro `split` uses keyword parameters (identified with the `$KEYS` command) to specify a file name prefix (`file`) and a component suffix set (`comps`, e.g., bhe, bhn, bhz) on the command line. A `pick` keyword is optional with a default. Further input data is obtained from the SAC file headers. If sufficient information exists in the headers for processing, the macro proceeds, unless `pick yes` indicates that the waveform should be re-picked. If picking is needed, `PLOTPK` starts a graphics interaction to define the start and end times of a waveform in the seismogram through picks of `A` and `F`.

Data is then provided to the program through the command line arguments and via standard input. The macro uses `SYSTEMCOMMAND` to pass the args to the program and to connect standard input to an input file written by the macro using the *echo* command and `SYSTEMCOMMAND`. The program uses the Fortran library subroutine `GETARG` to retrieve the command line inputs and uses Fortran I/O to read text from the standard input. To read header information, the program uses the SAC I/O library routine `RSAC1` to read the file without reading any data. Then the SAC I/O library routine `GETxHV` extracts information from the header. After preliminary checks to ensure that the headers contain event and station information, the program uses `RSAC1` again to read the data for each trace.

After the splitting analysis, the SAC I/O subroutine `WSAC0` is used to write out the original horizontal waveforms (a pair of traces) and the time-shifted waveforms (another pair). The program avoids using `NEWHDR` so that the old header information in the data traces is retained. Calls to `SETxHV` change the header variables that provide the macro with the estimated splitting parameters: ϕ and its uncertainty (`USER0`, `USER1`), δt and its uncertainty (`USER2` and `USER3`). They also return the estimated initial polarization ϕ_0 (`USER4`) and the degrees of freedom in the measurement window (`USER5` and `USER6`). The program writes these files using `WSAC0`. The misfit error surface (an example of a 3D SAC file – see Chapter 10) is similarly output, but the header variables `USER0` ... `USER6` are filled in with values used later in the `split` macro. Upon return from the analysis program, the `split` macro reads these various new SAC files and presents them for QC of the analysis. The particle motion plot (Fig. 11.4) is made with `PLOTPM` and `PLOT2` in different viewports of a composite plot. The misfit surface is displayed using `CONTOUR` to show the optimum value (ϕ and δt) and their joint uncertainty.

In addition to the Silver and Chan (1991) codes, SAC macros and codes to implement the stacking method of Restivo and Helffrich (1999) accompany this text. In this technique, high signal-to-noise ratio measurements are given more weight. Additionally, every individual measurement is scaled to a factor of $1/N$, with its back-azimuth defining a wedge of $\pm 10^\circ$ in which N observations fall. This compensates for the effects of over-represented back-azimuths in the uneven sampling of a station. The stacking procedure is implemented via the SAC macro `splstack`. The macro uses keyword parameters, with the obligatory `file` keyword indicating a file name that contains a list of splitting estimates to combine, one per line. Other keyword parameters are optional with suitable defaults. The macro passes the

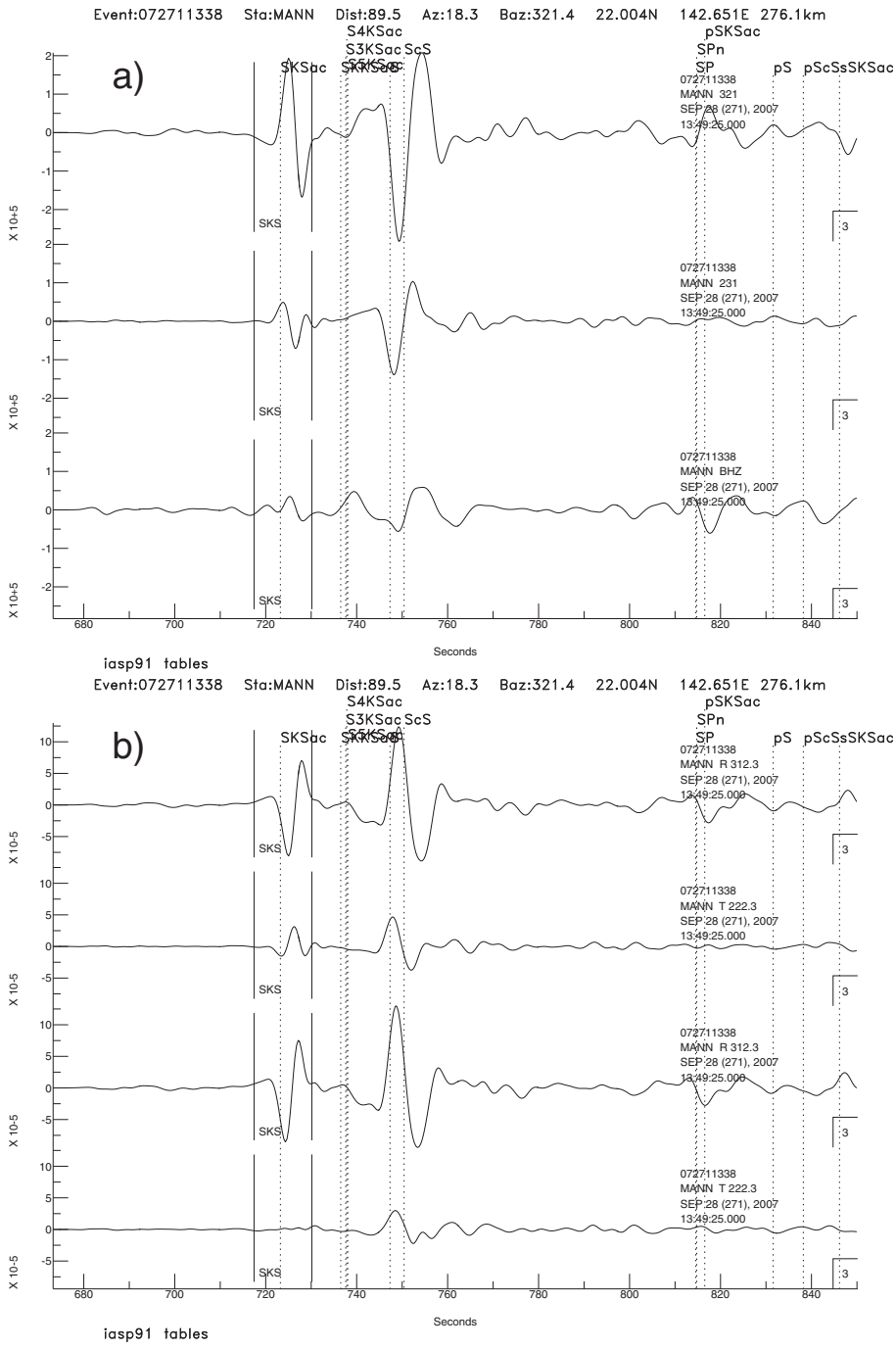


Figure 11.2 Shear wave splitting analysis example using data from station MANN in northern Hudson Bay, Canada (Bastow *et al.*, 2011). (*a*) Three-component broadband seismograms, cut around the *SKS* phase. Time window for splitting analysis indicated with labeled picks around the *SKS* arrival. (*b*) Top two traces are original radial and tangential component seismograms; bottom two traces are radial and tangential traces after correction for splitting. Note that, in the corrected seismogram, tangential component energy has been minimized.

input file to the auxiliary shell script `splitstack.sh` that invokes the program `splstacksac` to average the estimates. It then returns a 3D file to the macro to display by `CONTOUR`, with annotations added by `PLOTC`.

The program reads a list of file names and a command line arg, similar to the way that the splitting analysis program works. Each file header is read (using `RSAC1`), suitable information is loaded, and then data is read for stacking. A 3D file, written using `WSAC0`, is the result of the stack estimation. The stack-estimated ϕ and δt, their uncertainty and the joint degrees of freedom are returned in `USER1` ... `USER5` for macro use. The macro reads the stack result and presents it as a contoured file using the `CONTOUR` command with `PLOTC` annotations.

SAC implementation

To perform the shear wave splitting analysis on a set of three seismograms for station MANN in northern Hudson Bay, Canada (Bastow *et al.*, 2011), where files take the form MANN.133859.sks.bh[e,n,z], type the following:

```
SAC> window 1 x 0.05 0.80 y 0.05 0.74
SAC> m split file MANN.133859.sks comps bhe bhn bhz option e
```

The `split` macro uses SAC graphics to display the seismogram and allows the analyst to define a window with the keys *a* and *f* (to set the `A` and `F` file header variables) around the *SKS* phase (Fig. 11.2a). The program will then display a number of images to document the stages in the shear wave splitting analysis. Radial and tangential component seismograms are shown both before and after the splitting analysis (Fig. 11.2b). Note the energy on the tangential component of Figure 11.2b, minimized after correction.

Next, the windowed data is shown with focus on the fast and slow shear waves before and after correction (Fig. 11.3). Particle motion plots enable the analyst to establish that elliptical particle motion, diagnostic of shear wave splitting, is linearized after correction.

Finally, an uncertainty plot is displayed (Fig. 11.4), showing the results of the grid search over all possible values of ϕ and δt. The bold contour outlines the area of joint uncertainty in ϕ and δt, showing how good the measurement is – a smaller area corresponds to lower uncertainty. The splitting parameters resulting from methods such as Silver and Chan (1991) are generally shown as vectors on a map, with the orientation of the arrow paralleling ϕ, and the length of it proportional to δt (see e.g., figure 1 in Bastow *et al.*, 2011).

11.3 RECEIVER FUNCTION ANALYSIS

Overview

Receiver functions are time series computed from three-component seismic data that show the impulse response of Earth structure beneath a seismometer to an incoming plane wave (e.g., Langston, 1979). The P receiver function waveform is a composite of P-to-S converted waves and subsequent reverberations generated at impedance discontinuities such as the Moho (Fig. 11.5) and mantle transition zone.

Modeling the amplitude and timing of those reverberating waves can supply valuable constraints on the underlying geology and mantle structure. In many cases, the Earth structure can be approximated by a series of roughly horizontal layers. The arrivals generated by

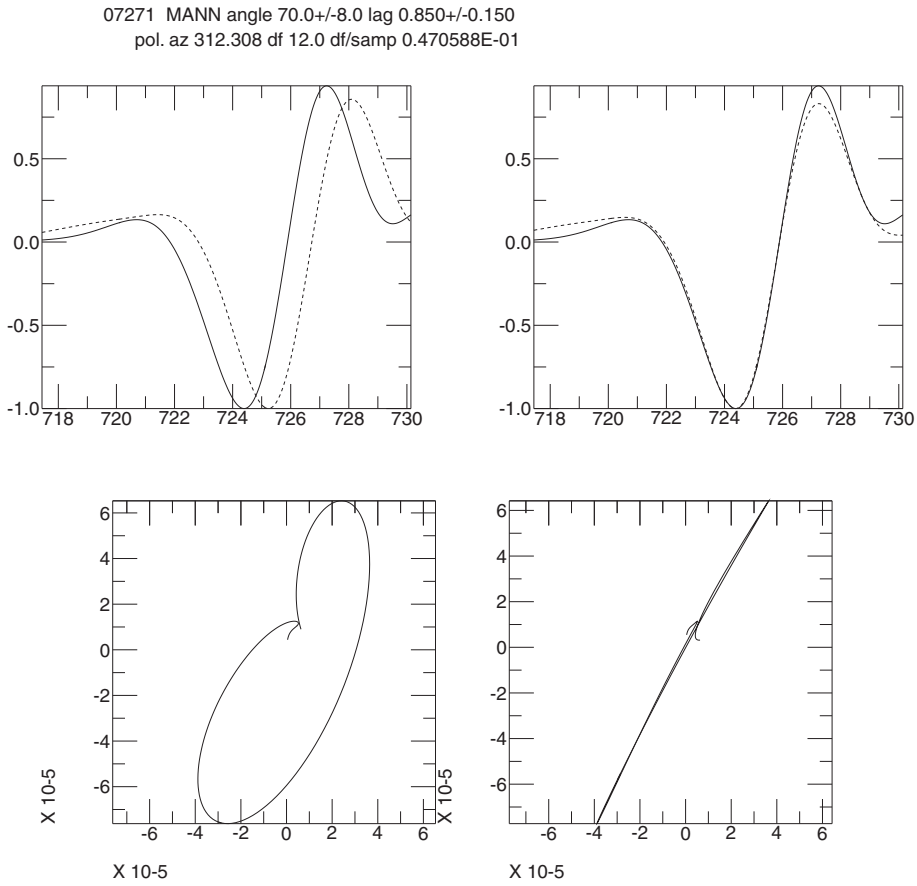


Figure 11.3 (*top*) Superposition of fast (ϕ direction) and slow components ($\phi + 90°$), uncorrected (*left*) and corrected (*right*). (bottom) Particle motion for slow components, uncorrected (*left*) and corrected (*right*). Corrections are made using estimated values of the splitting parameters, ϕ and δt, to assess their validity.

each sharp boundary (that is, sharp relative to the shortest wavelength in the observations) look something like the cartoon seismogram in Figure 11.5.

A radial receiver function can be computed using the source equalization procedure of Langston (1979) by deconvolving the vertical component seismogram from the radial (SV) component (and by following a similar procedure for the tangential component, SH). In the frequency domain, this can be written simply as

$$H(f) = R(f)/Z(f) \tag{11.1}$$

where f is the frequency, $Z(f)$ and $R(f)$ are the Fourier transforms of the vertical and radial seismograms, and $H(f)$ is the Fourier transform of the receiver function.

In the following section, we review briefly the theory behind receiver function analysis, for which we provide source code and operator instructions.

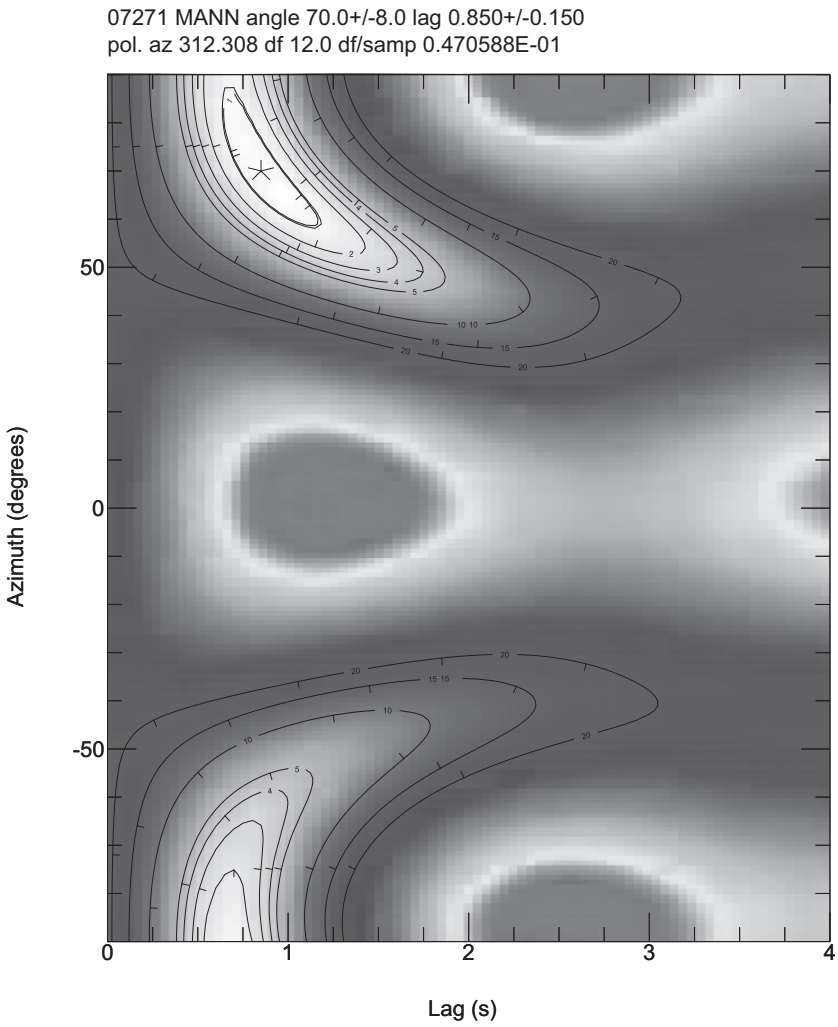


Figure 11.4 Results of the grid search over δt and ϕ. The optimum splitting parameters are shown by the star (top left). The 95% confidence region is shown (double contour), along with multiples of that contour level. (For color version, see Plates section.)

Estimation methodologies

Equation 11.1 is a relatively simple concept but implementation is difficult because of the instability of deconvolution. Thus, a number of different approaches to the computation of receiver functions has been developed over the years:

- water level deconvolution (e.g., Langston, 1979);
- deconvolution in the time domain by least squares (e.g., Abers *et al.*, 1995);
- iterative deconvolution in the time domain (e.g., Ligorrìa and Ammon, 1999);
- multi-taper frequency-domain cross-correlation receiver function (MTRF) (e.g., Park and Levin, 2000); MTRF is more resistant to noise so better for ocean island environments, for

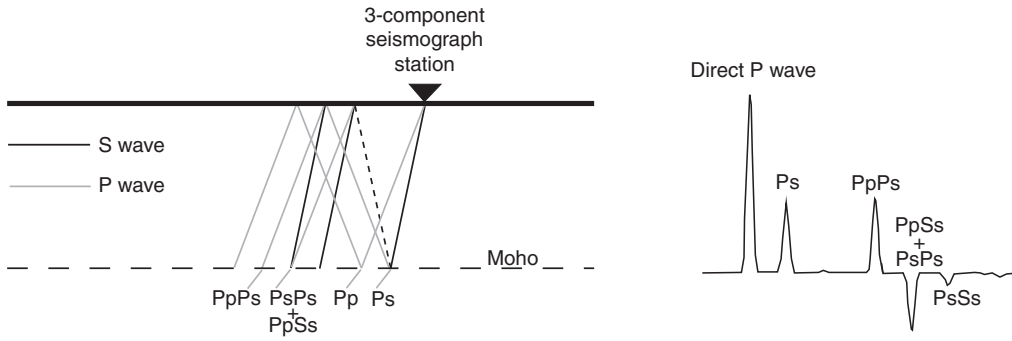


Figure 11.5 Receiver function analysis of Earth structure beneath a seismograph station. (*left*) When a P-wave is incident on a velocity discontinuity such as the Moho, it generates a shear wave at the interface. These arrivals, and their subsequent reverberations, can be isolated by receiver function analysis (*right*). Modified after Ammon (1991).

example, an advantage that is due to the use of multi-tapers to minimize spectral leakage and its frequency-dependent down-weighting in noisy portions of the spectrum.

Macro and auxiliary program design

The implementation of MTRF methodology in SAC is intuitively very similar to `split` code. The `mtrf` macro invokes SAC to display three-component seismogram data such that the analyst can define a measurement interval using picks for the `A` and `F` header variables. SAC rotates the horizontal component traces to the radial and tangential frames for the program. Macro keyword parameters provide user input, including a file name prefix, a component suffix set (e.g., bhe, bhn, bhz), a frequency-domain tapering limit (f_{max}) and phase information. The macro passes the information to the Fortran program *mtdecon* via command line args and the standard input. The program calls `GETARG` and the SAC library subroutine `RSAC1` to extract header information and trace data. `WSAC0` is used to output the results of the analysis (vertical, radial and tangential component receiver functions), which subsequently can be displayed in SAC for quality control (Fig. 11.6).

SAC implementation

To perform the receiver function analysis on a set of three seismograms for permanent station FRB in northern Hudson Bay, Canada, where file names take the form FRB.133859.p.bh[e,n,z], type the following:

```
SAC> m mtrf file FRB.133859.p comps bhe bhn bhz pick yes fmax 1
```

Here, `fmax` governs the frequency content of the resulting receiver function.

The `mtrf` macro uses graphical interactions (`PLOTPK`) to display the seismogram and allow the analyst to define a window through picks of the file header variables `A` and `F` beginning just before the first arriving P-wave phase and ending ~30 s into the P-wave coda

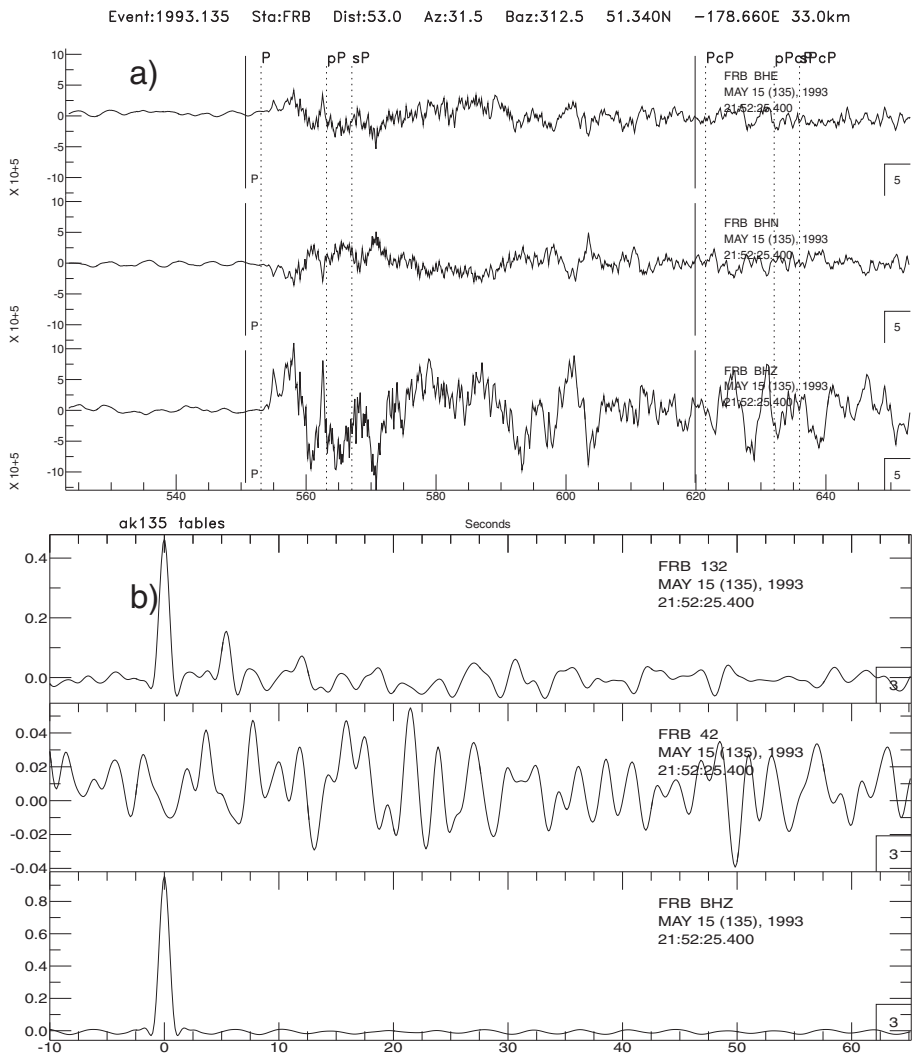


Figure 11.6 Example receiver function analysis using data from station FRB in northern Hudson Bay, Canada. (*a*) Three-component broadband seismograms, windowed around the P phase. (*b*) Radial (*top*) and tangential (*middle*) component receiver functions resulting from the extended-time MTRF method (Helffrich, 2006). Note the identifiable P-to-S conversion arriving at ~5.5 s, which likely developed at the Moho beneath the station.

(Fig. 11.6a). Then `mtrf` displays the results of the receiver function analysis for quality assurance. Figure 11.6b shows the resulting display.

H–κ analysis

After computation, receiver function data is routinely analyzed further by auxiliary SAC programs. One example is the H–κ stacking method of Zhu and Kanamori (2000), which estimates bulk crustal properties (crustal thickness and V_P/V_S ratio) from the receiver function

time series. This procedure utilizes SAC commands and subroutines for reading SAC-format data, writing output files and, finally, displaying output graphically.

Crustal thickness (H) and V_P/V_S ratio (κ) are known to trade off strongly (e.g., Ammon *et al.*, 1990; Zandt *et al.*, 1995). In an effort to reduce the ambiguity inherent in this trade-off, Zhu and Kanamori (2000) incorporated the later arriving crustal reverberations PpPs and PpSs + PsPs in a stacking procedure whereby the stacking itself transforms the time-domain receiver functions directly into objective function values in H–κ parameter space. This method, known as the H–κ stacking technique, is used here. The objective function for stacking is

$$s(H,\kappa) = \sum_{j=1}^{N} w_1 r_j(t_1[H,\kappa]) + w_2 r_j(t_2[H,\kappa]) - w_3 r_j(t_3[H,\kappa]) \tag{11.2}$$

where w_1, w_2, w_3 are weights; $r_j(t)$ are the receiver function amplitudes at the predicted arrival times of the direct P-to-S conversion Ps and subsequent reverberations PpPs and PsPs + PpSs, respectively, for the jth receiver function; and N is the number of receiver functions used. The predicted travel times t_i are given by Equations 11.3–11.5:

$$t_1[H,\kappa] = H\left[\sqrt{\frac{\kappa^2}{V_P^2} - p^2} - \sqrt{\frac{1}{V_P^2} - p^2}\right] \tag{11.3}$$

$$t_2[H,\kappa] = H\left[\sqrt{\frac{\kappa^2}{V_P^2} - p^2} + \sqrt{\frac{1}{V_P^2} - p^2}\right] \tag{11.4}$$

$$t_3[H,\kappa] = 2H\sqrt{\frac{\kappa^2}{V_P^2} - p^2} \tag{11.5}$$

where p is the ray parameter (s / km) and V_P is an assumed P-wave speed representing the reverberative interval.

The stacked objective function should attain its maximum value when H and κ are correct. Thus by performing a grid search for a range of plausible H and κ values, its maximum value can be determined (Zhu and Kanamori, 2000). For example, an H–κ grid search might range over $20 \le H \le 50$ km and $1.5 \le \kappa \le 2.3$. The H–κ method provides a better estimate of Moho depth and V_P/V_S ratio than a simple stacking of all receiver functions because it accounts for the receiver function dependence on ray parameter, upon which the objective function sensitively depends.

Once a number of radial receiver functions have been computed for a given seismograph station, the receiver functions can be viewed and stacked in SAC:

```
SAC> read *.nrfr
SAC> sss
SAC/SSS> tw -10 30; dw units degrees usedata
SAC/SSS> prs
SAC/SSS> cs all sum on
SAC/SSS> sumstack
```

This can often result in later reverberant phases such as PpPs and PsPs + PpSs (Fig. 11.5) emerging from below noise level. However, if earthquakes from variable angular distances from the seismograph station are used in the analysis, the expected travel times of Ps and its subsequent reverberant phases change (Equations 11.3–11.5). The controlling parameter is horizontal slowness p, rather than range.

Further analyses like H–κ depend on p, so associating slowness with the trace is beneficial. An auxiliary program *setrfslow* adds the horizontal slowness information to the headers of the radial receiver function files in units of seconds/degree.

```
ls *nrfr | setrfslow -set USER9 -units s/km -verbose
```

In this case, the variable is `USER9`. The following commands display the radial receiver function dependence on slowness:

```
SAC> sc cat doss
* Contents of macro "doss"
read *.nrfr
sss
do i from 1 to (status NFILES)
   changestack $i$ distance &$i$,user9&
enddo
timewindow -10 30 ; distancewindow usedata
prs
qs
SAC> m doss
```

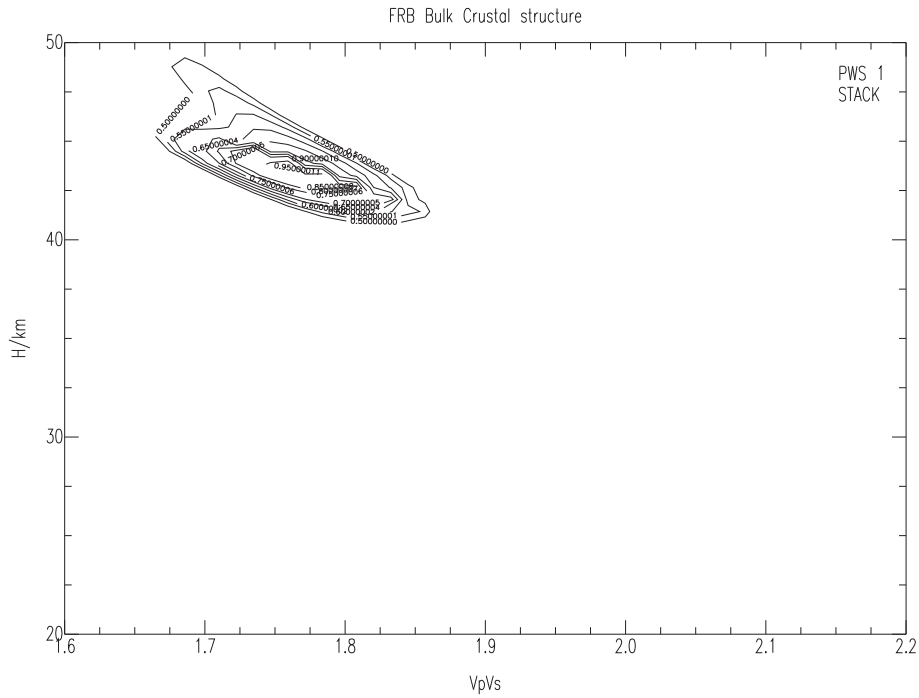


Figure 11.7 Illustration of the H–κ method using SAC graphics for permanent station FRB in Canada. In their receiver function study, Thompson *et al.* (2010) constrained bulk crustal properties for station FRB to be H = 43.5 km, V_P/V_S= 1.75, as is observed here.

In the directory of radial receiver functions, pipe the receiver functions into the H–κ stacking code. This will evaluate the objective function (Equation 11.2) for 50 crustal thickness (H) in the range 20–50 km, V_p/V_s in the range 1.6–2.1 with weights 0.5, 0.4 and 0.1.

```
ls *nrfr | \
   hk -vp 6.5 -h 20 50 50 -k 1.6 2.1 50 \
      -weight 0.5 0.4 0.1 -phaseweight 1 \
      -p USER9 -info HK.sac
```

Then, use SAC graphics to view the H–κ stack as follows (Fig. 11.7):

```
SAC> r HK.sac
SAC> zlevels range 0.5 1 increment 0.05
SAC> xlabel "VpVs"
SAC> ylabel "H / km"
SAC> title "FRB Bulk Crustal Structure"
SAC> zlabels on
SAC> contour
```

It is easy to envisage developing a macro to aid in the display of H–κ stacks that might even include annotating the plot with PLOTC, showing the peak using a crosshair and overlaying the contoured data onto a color background.

Appendix

A

Alphabetical list of SAC commands

absolutevalue	Takes the absolute value of each data point.
add	Adds a constant to each data point.
addf	Adds a set of data files to data in memory.
addstack	Adds a new file to the stack file list.
apk	Applies an automatic event picking algorithm.
arraymap	Produces a map of the array or "coarray" using all files in SAC memory.
axes	Controls the location of annotated axes.
bandpass	Applies an infinite-impulse-response (IIR) bandpass filter.
bandrej	Applies an IIR bandreject filter.
bbfk	Computes the broadband frequency–wavenumber (FK) spectral estimate, using all files in SAC memory.
beam	Computes the beam using all data files in SAC memory.
begindevices	Begins plotting to one or more graphics devices.
beginframe	Turns off automatic new frame actions between plots.
beginwindow	Begins plotting to a new graphics window.
benioff	Applies a Benioff filter to the data.
binoperr	Controls errors that can occur during binary file operations.
border	Controls the plotting of a border around plots.
break	Exits early from a sequence of repeated commands inside of a macro.
changestack	Changes properties of files currently in the stack file list.
chnhdr	Changes the values of selected header fields.
chpf	Closes the currently open HYPO pick file.
color	Controls color selection for color graphics devices.

comcor Controls SAC's command correction option.
comment Allows a comment in a SAC macro or on a SAC command line.
contour Produces contour plots of data in memory.
convert Converts data files from one format to another.
convolve Computes the convolution of a master signal with one or more other signals; computes the auto- and cross-convolution functions.
copyhdr Copies header variables from one file in memory to all others.
copyright Displays copyright.
cor Computes the correlation function.
correlate Computes the auto- and cross-correlation functions.
cut Defines how much of a data file is to be read.
cuterr Controls errors due to bad cut parameters.
cutim Cuts files in memory; can cut multiple segments from each file.
datagen Generates sample data files and stores them in memory.
decimate Decimates (downsamples) data, including an optional anti-aliasing finite-impulse-response (FIR) filter.
deletechannel Deletes one or more files from the file list.
deletestack Deletes one or more files from the stack file list.
deltacheck Changes the sampling rate checking option.
dif Differentiates data in memory.
distanceaxis Defines the record section plot distance axis parameters.
distancewindow Controls the distance window properties on subsequent record section plots.
div Divides each data point by a constant.
divf Divides data in memory by a set of data files.
divomega Performs integration in the frequency domain.
do Repeats a sequence of commands inside of a macro.
echo Controls echoing of input and output to the terminal.
else Conditionally processes a sequence of commands inside of a macro.
elseif Conditionally processes a sequence of commands inside of a macro.
enddevices Terminates one or more graphics devices.
enddo Ends a sequence of repeated commands inside of a macro.
endframe Resumes automatic new frame actions between plots.
endif Conditionally processes a sequence of commands inside of a macro.
endwindow Closes a graphics window.
envelope Computes the envelope function using a Hilbert transform.
erase Erases the graphics display area.
evaluate Evaluates simple arithmetic expressions.
exp Computes the exponential of each data point.
exp10 Computes the base 10 exponential (10**y) of each data point.
expressions Overview of expressions in SAC
fft Performs a discrete Fourier transform.
fileid Controls the file id display found on most SAC plots.
filenumber Controls the file number display found on most SAC plots.
filterdesign Produces a graphic display of a filter's digital versus analog characteristics for amplitude, phase and impulse response curves, and the group delay.
fir Applies an FIR filter.

floor Puts a minimum value on logarithmically scaled data.
funcgen Generates a function and stores it in memory.
getbb Gets (prints) values of blackboard variables.
getfhv Returns a real-valued header variable from the present SAC file.
getihv Returns an enumerated header variable from the present SAC file.
getkhv Returns a character-valued header variable from the present SAC file.
getlhv Returns a logical-valued header variable from the present SAC file.
getnhv Returns an integer-valued header variable from the present SAC file.
globalstack Sets global stack properties.
grayscale Produces grayscale or color images of data in memory.
grid Controls the plotting of grid lines in plots.
gtext Controls the quality and font of text used in plots.
hanning Applies a "Hanning" window to each data file.
help Displays information about SAC commands and features on the screen.
highpass Applies an IIR highpass filter.
hilbert Applies a Hilbert transform.
history Prints a list of the recently issued SAC commands
if Conditionally processes a sequence of commands inside of a macro.
ifft Performs an inverse discrete Fourier transform.
image Produces color sampled image plots of data in memory.
incrementstack Increments properties for files in the stack file list.
inicm Reinitializes all of SAC's common blocks.
installmacro Installs macro files in the global SAC macro directory.
int Performs integration using the trapezoidal or rectangular rule.
interpolate Interpolates evenly or unevenly spaced data to a new sampling rate.
keepam Keep amplitude component of spectral files (of either the AMPH or RLIM format) in SAC memory.
khronhite Applies a Khronhite filter to the data.
line Controls the linestyle selection in plots.
linefit Computes the best straight-line fit to the data in memory and writes the results to header blackboard variables.
linlin Turns on linear scaling for the x and y axes.
linlog Turns on linear scaling for x axis and logarithmic for y axis.
listhdr Lists the values of selected header fields.
liststack Lists the properties of the files in the stack file list.
load Load an external command (not available in MacSAC).
loadctable Allows the user to select a new color table for use in image plots.
log Takes the natural logarithm of each data point.
log10 Takes the base 10 logarithm of each data point.
loglab Controls labels on logarithmically scaled axes.
loglin Turns on logarithmic scaling for x axis and linear for y axis.
loglog Turns on logarithmic scaling for the x and y axes.
lowpass Applies an IIR lowpass filter.
macro Executes a SAC macro file and the startup/init commands when invoking SAC.

map Generate a Generic Mapping Tools (GMT) map with station/event symbols topography and station names using all the files in SAC memory and an optional event file specified on the command line.

markptp Measures and marks the maximum peak-to-peak amplitude of each signal within the measurement time window.

marktimes Marks files with travel times from a velocity set.

markvalue Searches for and marks values in a data file.

mem Calculates the spectral estimate using the maximum entropy method.

merge Merges (concatenates) a set of files to data in memory.

message Sends a message to the user's terminal.

mlm Calculates the spectral estimate using the maximum likelihood method.

mtw Determines the measurement time window for use in subsequent measurement commands.

mul Multiplies each data point by a constant.

mulf Multiplies a set of files by the data in memory.

mulomega Performs differentiation in the frequency domain.

news Prints current news concerning SAC.

null Sets the undefined sample value for plotting data.

oapf Opens an alphanumeric pick file.

ohpf Opens a HYPO formatted pick file.

pause Sends a message to the terminal and pauses.

pds Calculates the spectral estimate using the power density spectrum method.

phase Specifies phase names to include in record section plots.

picks Controls the display of time picks on most SAC plots.

plabel Defines general plot labels and their attributes.

plot Generates a single-trace single-window plot.

plot1 Generates a multi-trace multi-window plot.

plot2 Generates a multi-trace single-window (overlay) plot.

plotalpha Reads alphanumeric data from disk into memory and plots the data, optionally labeling each data point, to the current output device.

plotcor Plots the correlation function.

plotc Annotates SAC plots and creates figures using cursor.

plotdy Plots one or more data files versus another data file with vertical error bars around each data value.

plotpe Plots the root mean square (RMS) prediction error function.

plotpk Produces a plot for the picking of arrival times.

plotpm Generates a "particle-motion" plot of pairs of data files.

plotrecordsection Plots a record section of the files in the stack file list.

plotsp Plots spectral data in several different formats.

plotspe Plots the spectral estimate.

plotstack Plots the files in the stack file list.

plotxy Plots one or more data files versus another data file.

production Controls the production mode option.

qdp Controls the "quick and dirty plot" option.

qsacxml	Queries a SAC XML dataset into a program and returns summary information about it.
quantize	Converts continuous data into its quantized equivalent.
quit	Terminates SAC.
quitsub	Terminates the currently active subprocess.
read	Reads data from SAC, SEGY, GCF or MSEED data files on disk into memory.
readbbf	Reads a blackboard variable file into memory.
readcor	Reads a SAC file containing the correlation function.
readcss	Reads data files in CSS external format from disk into memory.
readerr	Controls errors that occur during the READ command.
readgse	Reads data files in GSE 2.0 format from disk into memory.
readhdr	Reads headers from SAC data files into memory.
readsp	Reads spectral files written by WRITESP and WRITESPE.
readtable	Reads alphanumeric data files in column format on disk into memory.
report	Informs the user about the current state of SAC.
reverse	Reverses the order of data points.
rglitches	Removes glitches and timing marks.
rmean	Removes the mean.
rms	Computes the RMS of the data within the measurement time window.
rotate	Rotates a pair of data components through an angle.
rq	Removes the seismic Q factor from spectral data.
rsac1	Reads evenly spaced files into a program.
rsac2	Reads unevenly spaced or spectral files into a program.
rsach	Reads the header from a SAC file into a program.
rsacxml	Reads a trace from an XML SAC dataset into a program.
rtrend	Removes the linear trend.
setbb	Sets (defines) values of blackboard variables.
setdevice	Defines a default graphics device to use in subsequent plots.
setfhv	Sets a real-valued header variable in the present SAC file.
setihv	Sets an enumerated header variable in the present SAC file.
setkhv	Sets a character-valued header variable in the present SAC file.
setlhv	Sets a logical header variable in the present SAC file.
setmacro	Defines a set of directories to search when executing a SAC macro file.
setnhv	Sets an integer-valued header variable in the present SAC file.
sgf	Controls the SAC Graphics File (SGF) device options.
smooth	Applies an arithmetic smoothing algorithm to the data.
spe	Activates the spectral estimation subprocess.
spectrogram	Calculates a spectrogram using all of the data in memory.
speid	Controls annotation of plots from the spectral estimation subprocess.
speread	Reads data from a SAC data file into memory.
sqr	Squares each data point.
sqrt	Takes the square root of each data point.
sss	Activates the signal stacking subprocess.

stretch	Stretches (upsamples) data, including an optional interpolating FIR filter.
sub	Subtracts a constant from each data point.
subf	Subtracts a set of data files from data in memory.
sumstack	Sums the files in the stack file list.
symbol	Controls the symbol plotting attributes.
synchronize	Synchronizes the reference times of all files in memory.
syntax	Prints basic information about SAC commands, the command summary and syntax.
systemcommand	Executes system commands from SAC.
taper	Applies a symmetric taper to each end of data.
ticks	Controls the location of tick marks on plots.
timeaxis	Controls the time axis properties on subsequent record section plots.
timewindow	Sets the time window limits for subsequent stack summations.
title	Defines the plot title and attributes.
trace	Controls the tracing of blackboard and header variables.
transcript	Controls output to the transcript files.
transfer	Performs deconvolution to remove an instrument response and convolution to apply another instrument response.
traveltime	Computes travel-time curves for pre-defined models or reads travel-time curves from ASCII text files.
tsize	Controls the text size attributes.
unsetbb	Unsets (deletes) blackboard variables.
unwrap	Computes amplitude and unwrapped phase.
utilities	Overview of utility programs for SGF file plots, header information and file format swapping.
velocitymodel	Sets stack velocity model parameters for computing dynamic delays.
velocityroset	Controls the placement of a velocity rosette on subsequent record section plots.
view	Changes the set of files that will be affected by SAC commands.
vspace	Changes the maximum size and shape of plots.
wait	Tells SAC whether or not to pause between plots or during text output.
while	Repeats a sequence of commands inside of a macro.
whiten	Flattens the spectrum of the input time series.
whpf	Writes auxiliary cards into the HYPO pick file.
width	Controls line widths for plotting.
wiener	Designs and applies an adaptive Wiener filter.
wild	Sets wildcard characters used in read commands to expand file lists.
window	Sets the location and shape of graphics windows.
write	Writes data in memory to disk.
writebbf	Writes a blackboard variable file to disk.
writecor	Writes a SAC file containing the correlation function.
writegse	Writes data files in GSE 2.0 format from memory to disk.
writehdr	Overwrites the headers on disk with those in memory.
writespe	Writes a SAC file containing the spectral estimate.
writesp	Writes spectral files to disk as “normal" data files.

writestack	Writes the stack summation to disk.
wsac0	Writes out a SAC file, using the current header variables, from a program.
wsac1	Writes out an evenly spaced SAC file from a program.
wsac2	Writes out an unevenly spaced SAC time series file from a program.
wsach	Writes the header in the program's memory to a SAC file.
wsacxml	Writes out an XML SAC dataset from a program.
xapiir	Subroutine access to SAC time series filtering capabilities.
xdiv	Controls the x axis division spacing.
xfudge	Changes the x axis "fudge factor."
xfull	Controls plotting of x axis full logarithmic decades.
xgrid	Controls plotting of grid lines in the x direction.
xlabel	Defines the x axis label and attributes.
xlim	Determines the plot limits for the x axis.
xlin	Turns on linear scaling for the x axis.
xlog	Turns on logarithimic scaling for the x axis.
xvport	Defines the viewport for the x axis.
ydiv	Controls the y axis division spacing.
yfudge	Changes the y axis "fudge factor."
yfull	Controls plotting of y axis full logarithmic decades.
ygrid	Controls plotting of grid lines in the y direction.
ylabel	Defines the y axis label and attributes.
ylim	Determines the plot limits for the y axis.
ylin	Turns on linear scaling for the y axis.
ylog	Turns on logarithimic scaling for the y axis.
yvport	Defines the viewport for the y axis.
zcolors	Controls the color display of contour lines.
zerostack	Zeroes or reinitializes the signal stack.
zlabels	Controls the labeling of contour lines with contour level values.
zlevels	Controls the contour line spacing in subsequent contour plots.
zlines	Controls the contour line styles in subsequent contour plots.
zticks	Controls the labeling of contour lines with directional tick marks.

Appendix

B

Keyword in context for SAC command descriptions

The entries in this table are sorted based on a selected set of keywords drawn from the command descriptions in SAC's online help information. The entries are sorted based on the keyword in the middle of each text column and its context (four words to either side of the keyword) is provided along with the name of the command at the left side of each entry. Equivalent information may also be obtained by the `HELP APROPOS KKKK` command where the keyword is `KKKK`.

absolutevalue	Takes the	absolute value of each...
beginframe	...off automatic new frame	actions between plots.
endframe	Resumes automatic new frame	actions between plots.
spe		Activates the spectral estimation...
sss		Activates the signal stacking...
quitsub	Terminates the currently	active subprocess.
wiener	Designs and applies an	adaptive Wiener filter.
addstack		Adds a new file...
add		Adds a constant to...
addf		Adds a set of...
view	...files that will be	affected by SAC commands.
apk	...an automatic event picking	algorithm.
smooth	Applies an arithmetic smoothing	algorithm to the data.
oapf	Opens an	alphanumeric pick file.
plotalpha	Reads	alphanumeric data from disk...
readtable	Reads	alphanumeric data files in...
keepam	...files (of either the	AMPH or RLIM format)...

filterdesign	... vs. analog characteristics for	amplitude, phase and impulse ...
keepam	Keeps	amplitude component of spectral ...
markptp	... maximum peak to peak	amplitude of each signal ...
unwrap	Computes	amplitude and unwrapped phase.
filterdesign	... a filter's digital versus	analog characteristics for amplitude, ...
rotate	... data components through an	angle.
axes	Controls the location of	annotated axes.
speid	Controls	annotation of plots from ...
decimate	... data, including an optional	anti-aliasing FIR filter.
erase	Erases the graphics display	area.
evaluate	Evaluates simple	arithmetic expressions.
smooth	Applies an	arithmetic smoothing algorithm to ...
arraymap	... a map of the	array or "coarray" using ...
plotpk	... for the picking of	arrival times.
traveltime	... reads travel-time curves from	ASCII text files.
plabel	... plot labels and their	attributes.
symbol	Controls the symbol plotting	attributes.
title	... the plot title and	attributes.
tsize	Controls the text size	attributes.
xlabel	... x axis label and	attributes.
ylabel	... y axis label and	attributes.
convolve	... other signals. Computes the	auto- and cross-convolution functions.
correlate	Computes the	auto- and cross-correlation ...
apk	Applies an	automatic event picking algorithm.
beginframe	Turns off	automatic new frame actions ...
endframe	Resumes	automatic new frame actions ...
whpf	Writes	auxiliary cards into the ...
axes	... the location of annotated	axes.
linlin	... the x and y	axes.
loglab	... labels on logarithmically scaled	axes.
loglog	... the x and y	axes.
distanceaxis	... record section plot distance	axis parameters.
linlog	... and logarithmic for y	axis.
linlog	... linear scaling for x	axis and logarithmic for ...
loglin	... and linear for y	axis.
loglin	... logarithmic scaling for x	axis and linear for ...
timeaxis	Controls the time	axis properties on subsequent ...
xdiv	Controls the x	axis division spacing.
xfudge	Changes the x	axis "fudge factor."
xfull	Controls plotting of x	axis full logarithmic decades.
xlabel	Defines the x	axis label and attributes.
xlim	... limits for the x	axis.
xlin	... scaling for the x	axis.
xlog	... scaling for the x	axis.
xvport	... viewport for the x	axis.
yvport	... viewport for the y	axis.
bandpass	Applies an IIR	bandpass filter.
bandrej	Applies an IIR	bandreject filter.

plotdy	... file with vertical error	bars around each data ...
exp10	Computes the	base 10 exponential (10**y) ...
log10	Takes the	base 10 logarithm of ...
syntax	Prints	basic information about SAC ...
beam	Computes the	beam using all data ...
benioff	Applies a	Benioff filter to the ...
binoperr	... that can occur during	binary file operations.
getbb	Gets (prints) values of	blackboard variables.
linefit	... the results to header	blackboard variables.
readbbf	Reads a	blackboard variable file into ...
setbb	Sets (defines) values of	blackboard variables.
trace	Controls the tracing of	blackboard and header variables.
unsetbb	Unsets (deletes)	blackboard variables.
writebbf	Writes a	blackboard variable file to ...
inicm	... all of SAC's common	blocks.
border	... the plotting of a	border around plots.
bbfk	Computes the	broadband frequency–wavenumber ...
whpf	Writes auxiliary	cards into the HYPO ...
filterdesign	... filter's digital versus analog	characteristics for: amplitude, phase, ...
wild	Sets wildcard	characters used in read ...
getkhv	Returns a	character-valued header variable from ...
setkhv	Sets a	character-valued header variable in ...
deltacheck	Changes the sampling rate	checking option.
chpf		Closes the currently open ...
endwindow		Closes a graphics window.
arraymap	... of the array or	"coarray" using all files ...
color	Controls	color selection for color ...
color	Controls color selection for	color graphics devices.
grayscale	Produces grayscale or	color images of data ...
image	Produces	color sampled image plots ...
loadctable	... to select a new	color table for use ...
zcolors	Controls the	color display of contour ...
readtable	... alphanumeric data files in	column format on disk ...
comcor	Controls SAC's	command correction option.
comment	... or on a SAC	command line.
load	Loads an external	command (not available in ...
map	... file specified on the	command line.
readerr	... occur during the READ	command.
syntax	... about SAC commands, the	command summary and syntax.
break	... a sequence of repeated	commands inside of a ...
do	Repeats a sequence of	commands inside of a ...
else	... processes a sequence of	commands inside of a ...
elseif	... processes a sequence of	commands inside of a ...
enddo	... a sequence of repeated	commands inside of a ...
endif	... processes a sequence of	commands inside of a ...
help	Displays information about SAC	commands and features on ...
history	... the recently issued SAC	commands.
if	... processes a sequence of	commands inside of a ...

macro	... file and the startup/init	commands when invoking SAC.
mtw	... use in subsequent measurement	commands.
syntax	... basic information about SAC	commands, the command summary ...
systemcommand	Executes system	commands from SAC.
view	... be affected by SAC	commands.
while	Repeats a sequence of	commands inside of a ...
wild	... characters used in read	commands to expand filelists.
comment	Allows a	comment in a SAC ...
keepam	Keeps amplitude	component of spectral files ...
rotate	... a pair of data	components through an angle.
merge	Merges	(concatenates) a set of ...
else		Conditionally processes a sequence ...
elseif		Conditionally processes a sequence ...
endif		Conditionally processes a sequence ...
if		Conditionally processes a sequence ...
add	Adds a	constant to each data ...
div	... data point by a	constant.
mul	... data point by a	constant.
sub	Subtracts a	constant from each data ...
quantize	Converts	continuous data into its ...
contour	Produces	contour plots of data ...
zcolors	... the color display of	contour lines.
zlabels	Controls the labeling of	contour lines with contour ...
zlabels	... of contour lines with	contour level values.
zlevels	Controls the	contour line spacing in ...
zlevels	... line spacing in subsequent	contour plots.
zlines	Controls the	contour line styles in subsequent ...
zlines	... contour linestyles in subsequent	contour plots.
zticks	Controls the labeling of	contour lines with directional ...
convolve	Computes the	convolution of a master ...
transfer	... an instrument response and	convolution to apply another ...
copyhdr		Copies header variables from ...
copyright	Displays	copyright.
comcor	Controls SAC's command	correction option.
cor	Computes the	correlation function.
correlate	... the auto- and cross-	correlation functions.
plotcor	Plots the	correlation function.
readcor	... SAC file containing the	correlation function.
writecor	... SAC file containing the	correlation function.
correlate	Computes the auto- and	cross-correlation functions.
convolve	... Computes the auto- and	cross-convolution functions.

plotxy	... data files versus another	data file.
quantize	Converts continuous	data into its quantized ...
read	Reads	data from SAC, SEGY, ...
read	... SEGY, GCF or MSEED	data files on disk ...
readcss	Reads	data files in CSS ...
readgse	Reads	data files in GSE ...
readhdr	Reads headers from SAC	data files into memory.
readtable	Reads alphanumeric	data files in column ...
reverse	Reverses the order of	data points.
rms	... mean square of the	data within the measurement ...
rotate	Rotates a pair of	data components through an ...
rq	... Q factor from spectral	data.
smooth	... smoothing algorithm to the	data.
spectrogram	... using all of the	data in memory.
speread	Reads	data from a SAC ...
speread	... data from a SAC	data file into memory.
sqr	Squares each	data point.
sqrt	... square root of each	data point.
stretch	Stretches (upsamples)	data, including an optional ...
sub	... a constant from each	data point.
subf	Subtracts a set of	data files from data ...
subf	... of data files from	data in memory.
taper	... to each end of	data.
write	Writes	data in memory to ...
writegse	Write	data files in GSE ...
writesp	... to disk as "normal"	data files.
qsacxml	Queries a SAC XML	dataset into a program ...
rsacxml	... from an XML SAC	dataset into a program.
wsacxml	... out an XML SAC	dataset from a program.
xfull	... x axis full logarithmic	decades.
yfull	... y axis full logarithmic	decades.
decimate		Decimates (downsamples) data, including ...
transfer	Performs	deconvolution to remove an ...
setdevice	Defines a	default graphics device to ...
setbb	Sets	(defines) values of blackboard ...
filterdesign	... curves, and the group	delay.
velocitymodel	... parameters for computing dynamic	delays.
deletechannel		Deletes one or more ...
deletestack		Deletes one or more ...
unsetbb	Unsets	(deletes) blackboard variables.
pds	... estimate using the power	density spectrum method.
wiener		Designs and applies an ...
plotalpha	... to the current output	device.
setdevice	Defines a default graphics	device to use in ...
sgf	... SAC Graphics File (SGF)	device options.
begindevices	... one or more graphics	devices.
color	... selection for color graphics	devices.

enddevices	... one or more graphics	devices.
plotsp	... spectral data in several	different formats.
dif		Differentiates data in memory.
mulomega	Performs	differentiation in the frequency ...
filterdesign	... display of a filter's	digital versus analog characteristics ...
xgrid	... lines in the x	direction.
ygrid	... lines in the y	direction.
zticks	... of contour lines with	directional tick marks.
setmacro	Defines a set of	directories to search when ...
installmacro	... the global SAC macro	directory.
qdp	Controls the "quick and	dirty plot" option.
fft	Performs a	discrete Fourier transform.
ifft	Performs an inverse	discrete Fourier transform.
plotalpha	Reads alphanumeric data from	disk into memory and ...
read	... MSEED data files on	disk into memory.
readcss	... CSS external format from	disk into memory.
readgse	... GSE 2.0 format from	disk into memory.
readtable	... in column format on	disk into memory.
write	... data in memory to	disk.
writebbf	... blackboard variable file to	disk.
writegse	... format from memory to	disk.
writehdr	Overwrites the headers on	disk with those in ...
writesp	Writes spectral files to	disk as "normal" data ...
writestack	... the stack summation to	disk.
copyright		Displays copyright.
erase	Erases the graphics	display area.
fileid	Controls the file id	display found on most ...
filenumber	Controls the file number	display found on most ...
filterdesign	Produces a graphic	display of a filter's ...
picks	Controls the	display of time picks ...
zcolors	Controls the color	display of contour lines.
help		Displays information about SAC ...
distanceaxis	... the record section plot	distance axis parameters.
distancewindow	Controls the	distance window properties on ...
div		Divides each data point ...
divf		Divides data in memory ...
xdiv	Controls the x axis	division spacing.
ydiv	Controls the y axis	division spacing.
divomega	... integration in the frequency	domain.
mulomega	... differentiation in the frequency	domain.
decimate	Decimates	(downsamples) data, including an ...
velocitymodel	... model parameters for computing	dynamic delays.
echo	Controls	echoing of input and ...
taper	... symmetric taper to each	end of data.
enddo		Ends a sequence of ...
mem	... estimate using the maximum	entropy method.

chpf	...currently open HYPO pick	file.
copyhdr	...header variables from one	file in memory to...
cut	...much of a data	file is to be...
cutim	...multiple segments from each	file.
deletechannel	...more files from the	file list.
deletestack	...files from the stack	file list.
fileid	Controls the	file id display found...
filenumber	Controls the	file number display found...
getfhv	...from the present SAC	file.
getkhv	...from the present SAC	file.
getlhv	...from the present SAC	file.
getnhv	...from the present SAC	file.
hanning	...window to each data	file.
incrementstack	...files in the stack	file list.
liststack	...files in the stack	file list.
macro	Executes a SAC macro	file and the startup/init...
map	...and an optional event	file specified on the...
markvalue	...values in a data	file.
oapf	Opens an alphanumeric pick	file.
ohpf	...a HYPO formatted pick	file.
plotdy	...files versus another data	file with vertical error...
plotrecordsection	...files in the stack	file list.
plotstack	...files in the stack	file list.
plotxy	...files versus another data	file.
readbbf	Reads a blackboard variable	file into memory.
readcor	Reads a SAC	file containing the correlation...
rsach	...header from a SAC	file into a program.
setfhv	...in the present SAC	file.
setihv	...in the present SAC	file.
setkhv	...in the present SAC	file.
setlhv	...in the present SAC	file.
setmacro	...executing a SAC macro	file.
setnhv	...in the present SAC	file.
sgf	Controls the SAC Graphics	File (SGF) device options.
speread	...from a SAC data	file into memory.
sumstack	...files in the stack	file list.
utilities	...plots, header information and	file format swapping.
utilities	...utility programs for SGF	file plots, header information...
whpf	...into the HYPO pick	file.
writebbf	Writes a blackboard variable	file to disk.
writecor	Writes a SAC	file containing the correlation...
writespe	Writes a SAC	file containing the spectral...
wsac0	Writes out a SAC	file, using the current...
wsac1	...out an evenly spaced SAC	file from a program.
wsac2	...unevenly spaced SAC time series	file from a program.
wsach	...memory to a SAC	file.
wild	...read commands to expand	filelists.
addf	...a set of data	files to data in...

arraymap	... or “coarray” using all	files in SAC memory.
bbfk	... spectral estimate, using all	files in SAC memory.
beam	... beam using all data	files in SAC memory.
changestack	Changes properties of	files currently in the ...
convert	Converts data	files from one format ...
cutim	Cuts	files in memory; can ...
datagen	Generates sample data	files and stores them ...
deletechannel	Deletes one or more	files from the file ...
deletestack	Deletes one or more	files from the stack ...
divf	... a set of data	files.
incrementstack	Increments properties for	files in the stack ...
installmacro	Installs macro	files in the global ...
keepam	... amplitude component of spectral	files (of either the ...
liststack	... the properties of the	files in the stack ...
map	... names using all the	files in SAC memory ...
marktimes	Marks	files with travel times ...
merge	... (concatenates) a set of	files to data in ...
mulf	Multiplies a set of	files by the data ...
plotdy	... one or more data	files versus another data ...
plotpm	... of pairs of data	files.
plotrecordsection	... record section of the	files in the stack ...
plotstack	Plots the	files in the stack ...
plotxy	... one or more data	files versus another data ...
read	... GCF or MSEED data	files on disk into ...
readcss	Reads data	files in CSS external ...
readgse	Reads data	files in GSE 2.0 ...
readhdr	... headers from SAC data	files into memory.
readsp	Reads spectral	files written by WRITESP ...
readtable	Reads alphanumeric data	files in column format ...
rsac1	Reads evenly spaced	files into a program.
rsac2	Reads unevenly spaced or spectral	files into a program.
subf	... a set of data	files from data in ...
sumstack	Sums the	files in the stack ...
synchronize	... reference times of all	files in memory.
transcript	... output to the transcript	files.
traveltime	... curves from ASCII text	files.
view	Changes the set of	files that will be ...
writegse	Writes data	files in GSE 2.0 ...
writesp	Writes spectral	files to disk as ...
writesp	... disk as “normal” data	files.
bandpass	Applies an IIR bandpass	filter.
bandrej	Applies an IIR bandreject	filter.
benioff	Applies a Benioff	filter to the data.
decimate	... an optional anti-aliasing FIR	filter.
fir	Applies a finite-impulse-response	filter.
highpass	Applies an IIR highpass	filter.
khronhite	Applies a Khronhite	filter to the data.
lowpass	Applies an IIR lowpass	filter.

stretch	... an optional interpolating FIR	filter.
wiener	... applies an adaptive Wiener	filter.
xapiir	... to SAC time series	filtering capabilities.
filterdesign	... graphic display of a	filter's digital versus analog ...
fir	Applies a	finite-impulse-response filter.
decimate	... including an optional anti-aliasing	FIR filter.
stretch	... including an optional interpolating	FIR filter.
linefit	... the best straight-line	fit to the data ...
bbfk	... broadband frequency–wavenumber	(FK) spectral estimate, using ...
whiten		Flattens the spectrum of ...
gtext	Controls the quality and	font of text used ...
convert	... data files from one	format to another.
keepam	... the AMPH or RLIM	format) in SAC memory.
readcss	... files in CSS external	format from disk into ...
readgse	... files in GSE 2.0	format from disk into ...
readtable	... data files in column	format on disk into ...
utilities	... header information and file	format swapping.
writegse	... files in GSE 2.0	format from memory to ...
plotsp	... data in several different	formats.
ohpf	Opens a HYPO	formatted pick file.
fileid	... the file id display	found on most SAC ...
filenumber	... the file number display	found on most SAC ...
fft	Performs a discrete	Fourier transform.
ifft	Performs an inverse discrete	Fourier transform.
beginframe	Turns off automatic new	frame actions between plots.
endframe	Resumes automatic new	frame actions between plots.
bbfk	Computes the broadband	frequency–wavenumber (FK) ...
divomega	Performs integration in the	frequency domain.
mulomega	Performs differentiation in the	frequency domain.
xfudge	Changes the x axis	"fudge factor."
yfudge	Changes the y axis	"fudge factor."
cor	Computes the correlation	function.
envelope	Computes the envelope	function using a Hilbert ...
funcgen	Generates a	function and stores it ...
plotcor	Plots the correlation	function.
plotpe	... the RMS prediction error	function.
readcor	... file containing the correlation	function.
writecor	... file containing the correlation	function.
convolve	... the auto- and cross-convolution	functions.
correlate	... auto- and cross-correlation	functions.
read	... data from SAC, SEGY,	GCF or MSEED data ...
plabel	Defines	general plot labels and ...
map	Generate a GMT	(Generic Mapping Tools) map ...
getbb		Gets (prints) values of ...
rglitches	Removes	glitches and timing marks.
globalstack	Sets	global stack properties.
installmacro	... macro files in the	global SAC macro directory.
map	Generate a	GMT (Generic Mapping Tools) ...

filterdesign	Produces a	graphic display of a . . .
begindevices	. . . to one or more	graphics devices.
beginwindow	. . . plotting to a new	graphics window.
color	. . . color selection for color	graphics devices.
enddevices	Terminates one or more	graphics devices.
endwindow	Closes a	graphics window.
erase	Erases the	graphics display area.
setdevice	Defines a default	graphics device to use . . .
sgf	Controls the SAC	Graphics File (SGF) device . . .
window	. . . location and shape of	graphics windows.
grayscale	Produces	grayscale or color images . . .
grid	Controls the plotting of	grid lines in plots.
xgrid	Controls plotting of	grid lines in the . . .
ygrid	Controls plotting of	grid lines in the . . .
filterdesign	. . . response curves, and the	group delay.
readgse	Reads data files in	GSE 2.0 format from . . .
writegse	Writes data files in	GSE 2.0 format from . . .
hanning	Applies a	“Hanning” window to each . . .
chnhdr	. . . the values of selected	header fields.
copyhdr	Copies	header variables from one . . .
getfhv	Returns a real-valued	header variable from the . . .
getihv	Returns an enumerated	header variable from the . . .
getkhv	Returns a character-valued	header variable from the . . .
getlhv	Returns a logical-valued	header variable from the . . .
getnhv	Returns an integer-valued	header variable from the . . .
linefit	. . . writes the results to	header blackboard variables.
listhdr	. . . the values of selected	header fields.
rsach	Reads the	header from a SAC . . .
setfhv	Sets a real-valued	header variable in the . . .
setihv	Sets an enumerated	header variable in the . . .
setkhv	Sets a character-valued	header variable in the . . .
setlhv	Sets a logical	header variable in the . . .
setnhv	Sets an integer-valued	header variable in the . . .
trace	. . . tracing of blackboard and	header variables.
utilities	. . . for SGF file plots,	header information and file . . .
wsac0	. . . file, using the current	header variables, from a . . .
wsach	Writes the	header in the program’s . . .
readhdr	Reads	headers from SAC data . . .
writehdr	Overwrites the	headers on disk with . . .
highpass	Applies an IIR	highpass filter.
envelope	. . . envelope function using a	Hilbert transform.
hilbert	Applies a	Hilbert transform.
chpf	Closes the currently open	HYPO pick file.
ohpf	Opens a	HYPO formatted pick file.
whpf	. . . auxiliary cards into the	HYPO pick file.
fileid	Controls the file	id display found on . . .
bandpass	Applies an	IIR bandpass filter.
bandrej	Applies an	IIR bandreject filter.

highpass Applies an IIR highpass filter.
lowpass Applies an IIR lowpass filter.
image Produces color sampled image plots of data . . .
loadctable . . . table for use in image plots.
grayscale Produces grayscale or color images of data in . . .
filterdesign . . . for amplitude, phase and impulse response curves, and . . .
incrementstack Increments properties for files . . .
help Displays information about SAC commands . . .
qsacxml . . . program and returns summary information about it.
syntax Prints basic information about SAC commands, . . .
utilities . . . SGF file plots, header information and file format . . .
report Informs the user about . . .
echo Controls echoing of input and output to . . .
whiten . . . the spectrum of the input time series.
transfer . . . convolution to apply another instrument response.
transfer . . . deconvolution to remove an instrument response and convolution . . .
getnhv Returns an integer-valued header variable from . . .
setnhv Sets an integer-valued header variable in . . .
divomega Performs integration in the frequency . . .
int Performs integration using the trapezoidal . . .
interpolate Interpolates evenly or unevenly . . .
stretch . . . data, including an optional interpolating FIR filter.
ifft Performs an inverse discrete Fourier transform.
macro . . . the startup/init commands when invoking SAC.
history . . . list of the recently issued SAC commands.
khronhite Applies a Khronhite filter to the . . .
xlabel Defines the x axis label and attributes.
ylabel Defines the y axis label and attributes.
plotalpha . . . plots the data, optionally labeling each data point, . . .
zlabels Controls the labeling of contour lines . . .
zticks Controls the labeling of contour lines . . .
loglab Controls labels on logarithmically scaled . . .
plabel Defines general plot labels and their attributes.
zlabels . . . contour lines with contour level values.
mlm . . . estimate using the maximum likelihood method.
timewindow Sets the time window limits for subsequent stack . . .
xlim Determines the plot limits for the x . . .
ylim Determines the plot limits for the y . . .
comment . . . on a SAC command line.
linefit Computes the best straight line fit to the . . .
map . . . specified on the command line.
width Controls line widths for plotting.
zlevels Controls the contour line spacing in subsequent . . .
linlin Turns on linear scaling for the . . .

linlog	Turns on	linear scaling for x...
loglin	...for x axis and	linear for y axis.
rtrend	Removes the	linear trend.
xlin	Turns on	linear scaling for the...
ylin	Turns on	linear scaling for the...
grid	...the plotting of grid	lines in plots.
xgrid	Controls plotting of grid	lines in the x...
ygrid	Controls plotting of grid	lines in the y...
zcolors	...color display of contour	lines.
zlabels	...the labeling of contour	lines with contour level...
zticks	...the labeling of contour	lines with directional tick...
line	Controls the	line style selection in plots.
zlines	Controls the contour	line styles in subsequent contour...
addstack	...to the stack file	list.
changestack	...in the stack file	list.
deletechannel	...files from the file	list.
deletestack	...from the stack file	list.
history	prints a	list of the recently...
incrementstack	...in the stack file	list.
liststack	...in the stack file	list.
plotrecordsection	...in the stack file	list.
plotstack	...in the stack file	list.
sumstack	...in the stack file	list.
listhdr		Lists the values of...
liststack		Lists the properties of...
load		Load an external command...
axes	Controls the	location of annotated axes.
ticks	Controls the	location of tick marks...
window	Sets the	location and shape of...
xlog	Turns on	logarithimic scaling for the...
ylog	Turns on	logarithimic scaling for the...
log	Takes the natural	logarithm of each data...
log10	Takes the base 10	logarithm of each data...
linlog	...for x axis and	logarithmic for y axis.
loglin	Turns on	logarithmic scaling for x...
loglog	Turns on	logarithmic scaling for the...
xfull	...of x axis full	logarithmic decades.
yfull	...of y axis full	logarithmic decades.
floor	...a minimum value on	logarithmically scaled data.
loglab	Controls labels on	logarithmically scaled axes.
setlhv	Sets a	logical header variable in...
getlhv	Returns a	logical-valued header variable from...
lowpass	Applies an IIR	lowpass filter.
break	...commands inside of a	macro.
comment	...comment in a SAC	macro or on a...
do	...commands inside of a	macro.
else	...commands inside of a	macro.

elseif . . . commands inside of a macro.
enddo . . . commands inside of a macro.
endif . . . commands inside of a macro.
if . . . commands inside of a macro.
installmacro Installs macro files in the . . .
installmacro . . . in the global SAC macro directory.
macro Executes a SAC macro file and the . . .
setmacro . . . when executing a SAC macro file.
while . . . commands inside of a macro.
load . . . command (not available in MacSAC).
arraymap Produces a map of the array . . .
map . . . GMT (Generic Mapping Tools) map with station/event symbols . . .
map Generate a GMT (Generic Mapping Tools) map with . . .
markptp Measures and marks the maximum peak . . .
marktimes Marks files with travel . . .
markvalue Searches for and marks values in a . . .
rglitches Removes glitches and timing marks.
ticks . . . the location of tick marks on plots.
zticks . . . lines with directional tick marks.
convolve . . . the convolution of a master signal with one . . .
markptp Measures and marks the maximum peak to peak . . .
mem . . . spectral estimate using the maximum entropy method.
mlm . . . spectral estimate using the maximum likelihood method.
vspace Changes the maximum size and shape . . .
rmean Removes the mean.
rms Computes the root mean square of the . . .
markptp . . . each signal within the measurement time window.
mtw Determines the measurement time window for . . .
mtw . . . for use in subsequent measurement commands.
rms . . . the data within the measurement time window.
markptp Measures and marks the . . .
cutim Cuts files in memory; can cut multiple . . .
merge Merges (concatenates) a set . . .
message Sends a message to the user's . . .
pause Sends a message to the terminal . . .
mem . . . using the maximum entropy method.
mlm . . . using the maximum likelihood method.
pds . . . the power density spectrum method.
floor Puts a minimum value on logarithmically . . .
production Controls the production mode option.
velocitymodel Sets stack velocity model parameters for computing . . .
traveltime . . . travel-time curves for pre-defined models or reads travel-time . . .
read . . . SAC, SEGY, GCF or MSEED data files on . . .
cutim . . . in memory; can cut multiple segments from each . . .
mul Multiplies each data point . . .
mulf Multiplies a set of . . .
plot1 Generates a multi-trace multi-window plot.
plot2 Generates a multi-trace single-window (overlay) plot.

plot1	Generates a multi-trace	multi-window plot.
map	. . . symbols topography and station	names using all the . . .
phase	Specifies phase	names to include in . . .
log	Takes the	natural logarithm of each . . .
news	Prints current	news concerning SAC.
writesp	. . . files to disk as	“normal” data files.
load	Loads an external command	(not available in MacSAC).
filenumber	Controls the file	number display found on . . .
keepam	. . . component of spectral files	(of either the AMPH . . .
chpf	Closes the currently	open HYPO pick file.
oapf		Opens an alphanumeric pick . . .
ohpf		Opens a HYPO formatted . . .
binoperr	. . . occur during binary file	operations.
comcor	Controls SAC’s command correction	option.
deltacheck	. . . the sampling rate checking	option.
production	Controls the production mode	option.
qdp	. . . “quick and dirty plot”	option.
plotalpha	. . . and plots the data,	optionally labeling each data . . .
sgf	. . . Graphics File (SGF) device	options.
reverse	Reverses the	order of data points.
echo	. . . echoing of input and	output to the terminal.
plotalpha	. . . point, to the current	output device.
transcript	Controls	output to the transcript . . .
wait	. . . plots or during text	output.
plot2	Generates a multi-trace single-window	(overlay) plot.
expressions		Overview of expressions in . . .
utilities		Overview of utility programs . . .
writehdr		Overwrites the headers on . . .
rotate	Rotates a	pair of data components . . .
plotpm	. . . a “particle-motion” plot of	pairs of data files.
cuterr	. . . due to bad cut	parameters.
distanceaxis	. . . section plot distance axis	parameters.
velocitymodel	Sets stack velocity model	parameters for computing dynamic . . .
plotpm	Generates a	“particle-motion” plot of pairs . . .
wait	. . . whether or not to	pause between plots or . . .
pause	. . . to the terminal and	pauses.
markptp	. . . and marks the maximum	peak-to-peak amplitude . . .
markptp	. . . the maximum peak to	peak amplitude of each . . .
filterdesign	. . . analog characteristics for amplitude,	phase and impulse response . . .
phase	Specifies	phase names to include . . .
unwrap	Computes amplitude and unwrapped	phase.
chpf	. . . the currently open HYPO	pick file.
oapf	Opens an alphanumeric	pick file.
ohpf	Opens a HYPO formatted	pick file.
whpf	. . . cards into the HYPO	pick file.
apk	Applies an automatic event	picking algorithm.
plotpk	. . . a plot for the	picking of arrival times.

picks	... the display of time	picks on most SAC ...
velocityroset	Controls the	placement of a velocity ...
distanceaxis	Defines the record section	plot distance axis parameters.
plabel	Defines general	plot labels and their ...
plot	Generates a single-trace single-window	plot.
plot1	Generates a multi-trace multi-window	plot.
plot2	... a multi-trace single-window (overlay)	plot.
plotpk	Produces a	plot for the picking ...
plotpm	Generates a “particle-motion”	plot of pairs of ...
qdp	... the “quick and dirty	plot” option.
title	Defines the	plot title and attributes.
xlim	Determines the	plot limits for the ...
ylim	Determines the	plot limits for the ...
beginframe	... new frame actions between	plots.
border	... of a border around	plots.
contour	Produces contour	plots of data in ...
distancewindow	... on subsequent record section	plots.
endframe	... new frame actions between	plots.
fileid	... found on most SAC	plots.
filenumber	... found on most SAC	plots.
grid	... of grid lines in	plots.
gtext	... of text used in	plots.
image	Produces color sampled image	plots of data in ...
line	... the line style selection in	plots.
loadctable	... for use in image	plots.
phase	... include in record section	plots.
picks	... picks on most SAC	plots.
plotalpha	... disk into memory and	plots the data, optionally ...
plotc	Annotates SAC	plots and creates figures ...
plotcor		Plots the correlation function.
plotdy		Plots one or more ...
plotpe		Plots the RMS prediction ...
plotrecordsection		Plots a record section ...
plotsp		Plots spectral data in ...
plotspe		Plots the spectral estimate.
plotstack		Plots the files in ...
plotxy		Plots one or more ...
setdevice	... to use in subsequent	plots.
speid	Controls annotation of	plots from the spectral ...
ticks	... of tick marks on	plots.
timeaxis	... on subsequent record section	plots.
utilities	... programs for SGF file	plots, header information and ...
velocityroset	... on subsequent record section	plots.
vspace	... size and shape of	plots.
wait	... not to pause between	plots or during text ...
zlevels	... spacing in subsequent contour	plots.
zlines	... line styles in subsequent contour	plots.

begindevices	Begins	plotting to one or . . .
beginwindow	Begins	plotting to a new . . .
border	Controls the	plotting of a border . . .
grid	Controls the	plotting of grid lines . . .
null	. . . undefined sample value for	plotting data.
symbol	Controls the symbol	plotting attributes.
width	Controls line widths for	plotting.
xfull	Controls	plotting of x axis . . .
xgrid	Controls	plotting of grid lines . . .
yfull	Controls	plotting of y axis . . .
ygrid	Controls	plotting of grid lines . . .
plotalpha	. . . optionally labeling each data	point, to the current . . .
pds	. . . spectral estimate using the	power density spectrum method.
traveltime	Computes travel-time curves for	pre-defined models or reads . . .
plotpe	Plots the RMS	prediction error function.
getbb	Gets	(prints) values of blackboard . . .
history		Prints a list of . . .
news		Prints current news concerning . . .
syntax		Prints basic information about . . .
production	Controls the	production mode option.
production	Controls the	production mode option.
qsacxml	. . . XML dataset into a	program and returns summary . . .
rsac1	. . . evenly spaced files into a	program.
rsac2	. . . spectral files into a	program.
rsach	. . . SAC file into a	program.
rsacxml	. . . SAC dataset into a	program.
wsac0	. . . header variables, from a	program.
wsac1	. . . SAC file from a	program.
wsac2	. . . series file from a	program.
wsacxml	. . . SAC dataset from a	program.
utilities	Overview of utility	programs for SGF file . . .
wsach	. . . the header in the	program's memory to a . . .
changestack	Changes	properties of files currently . . .
distancewindow	Controls the distance window	properties on subsequent record . . .
globalstack	Sets global stack	properties.
incrementstack	Increments	properties for files in . . .
liststack	Lists the	properties of the files . . .
timeaxis	Controls the time axis	properties on subsequent record . . .
rq	Removes the seismic	Q factor from spectral . . .
gtext	Controls the	quality and font of . . .
quantize	. . . continuous data into its	quantized equivalent.
qsacxml		Queries a SAC XML . . .
qdp	Controls the	"quick and dirty plot" . . .
deltacheck	Changes the sampling	rate checking option.
interpolate	. . . to a new sampling	rate.
cut	. . . file is to be	read.
readcss		Reads data files in . . .
readerr	. . . that occur during the	READ command.

loadctable	Allows the	user to select a . . .
report	Informs the	user about the current . . .
message	. . . a message to the	user's terminal.
utilities	Overview of	utility programs for SGF . . .
absolutevalue	Takes the absolute	value of each data . . .
floor	Puts a minimum	value on logarithmically scaled . . .
null	Sets the undefined sample	value for plotting data.
plotdy	. . . bars around each data	value.
chnhdr	Changes the	values of selected header . . .
getbb	Gets (prints)	values of blackboard variables.
listhdr	Lists the	values of selected header . . .
markvalue	Searches for and marks	values in a data . . .
setbb	Sets (defines)	values of blackboard variables.
zlabels	. . . lines with contour level	values.
getfhv	Returns a real-valued header	variable from the present . . .
getihv	Returns an enumerated header	variable from the present . . .
getkhv	Returns a character-valued header	variable from the present . . .
getlhv	Returns a logical-valued header	variable from the present . . .
getnhv	Returns an integer-valued header	variable from the present . . .
readbbf	Reads a blackboard	variable file into memory.
setfhv	Sets a real-valued header	variable in the present . . .
setihv	Sets an enumerated header	variable in the present . . .
setkhv	Sets a character-valued header	variable in the present . . .
setlhv	Sets a logical header	variable in the present . . .
setnhv	Sets an integer-valued header	variable in the present . . .
writebbf	Writes a blackboard	variable file to disk.
copyhdr	Copies header	variables from one file . . .
getbb	. . . (prints) values of blackboard	variables.
linefit	. . . results to header blackboard	variables.
setbb	. . . (defines) values of blackboard	variables.
trace	. . . of blackboard and header	variables.
unsetbb	Unsets (deletes) blackboard	variables.
wsac0	. . . using the current header	variables, from a program.
marktimes	. . . travel times from a	velocity set.
velocitymodel	Sets stack	velocity model parameters for . . .
velocityroset	. . . the placement of a	velocity rosette on subsequent . . .
plotdy	. . . another data file with	vertical error bars around . . .
xvport	Defines the	viewport for the x . . .
yvport	Defines the	viewport for the y . . .
filterdesign	. . . of a filter's digital	versus analog characteristics for . . .
bbfk	. . . the broadband frequency–	wavenumber (FK) spectral estimate, . . .
width	Controls line	widths for plotting.
wiener	. . . and applies an adaptive	Wiener filter.
wild	Sets	wildcard characters used in . . .
beginwindow	. . . to a new graphics	window.
distancewindow	Controls the distance	window properties on subsequent . . .
endwindow	Closes a graphics	window.

References

Abers, G.A., Hu, X., and Sykes, L.R. 1995. Source scaling of earthquakes in the Shumagin region, Alaska: time-domain inversions of regional waveforms. *Geophys. J. Int.*, **123**(1), 41–58.

Aki, K., and Richards, P.G. 1980. *Quantitative Seismology – Theory and Methods*. San Francisco: W.H. Freeman.

Ammon, C.J. 1991. The isolation of receiver effects from teleseismic *P* waveforms. *Bull. Seism. Soc. Am.*, **81**(6), 2504–2510.

Ammon, C.J., Randall, G.E., and Zandt, G. 1990. On the nonuniqueness of receiver function inversions. *J. Geophys. Res.*, **95**(B10), 15303–15318.

Anderson, J., Farrell, W., Garcia, K., Given, J., and Swanger, H. 1990. *CSS Version 3 Database: Schema Reference Manual*. Tech. Rep. C90-01. Science Applications International Corporation.

Barry, K.M., Cavers, D.A., and Kneale, C.W. 1975. Recommended standards for digital tape formats. *Geophysics*, **40**, 344–352.

Bastow, I.D., Owens, T.J., Helffrich, G., and Knapp, J.H. 2007. Spatial and temporal constraints on sources of seismic anisotropy: evidence from the Scottish highlands. *Geophys. Res. Lett.*, **34**(5), doi:10.1029/2006GL028911.

Bastow, I.D., Thompson, D.A., Wookey, J., Kendall, J., Helffrich, G., Snyder, D., Eaton, D., and Darbyshire, F. 2011. Precambrian plate tectonics: seismic evidence from northern Hudson Bay, Canada. *Geology*, **39**(1), 91–94.

Blackman, D., and Kendall, J.M. 1997. Sensitivity of teleseismic body waves to mineral texture and melt in the mantle beneath a mid–ocean ridge. *Phil. Trans. R. Soc. Lond.*, **355**, 217–231.

Bokelmann, G.H.R., and Silver, P.G. 2002. Shear stress at the base of shield lithosphere. *Geophys. Res. Lett.*, **29**(23), doi:10.1029/2002GL015925.

Burg, J.P. 1972. A new analysis technique for time series data. In: Childers, D.G. (ed.), *Modern Spectrum Analysis*. New York: IEEE Press. Originally published in NATO Advanced Study Institute: Signal processing with emphasis on underwater acoustics, Enschede, the Netherlands, 12–23 August 1968.

Crotwell, H.P., Owens, T.J., and Ritsema, J. 1999. The TauP toolkit: flexible seismic travel-time and ray-path utilities. *Seism. Res. Lett.*, **70**, 154–160.

Daubechies, I. 1992. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 61. Philadelphia, PA: SIAM.

Di Leo, J.F., Wookey, J., Hammond, J.O.S., Kendall, J.M., Kaneshima, S., Inoue, H., Yamashina, T., and Harjadi, P. 2012. Deformation and mantle flow beneath the Sangihe subduction zone from seismic anisotropy. *Phys. Earth Planet. Inter.*, **194**, 38–54.

Efron, B. 1982. *The Jackknife, the Bootstrap and Other Resampling Plans*. CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 38. Philadelphia, PA: SIAM.

Fouch, M.J., Fischer, A.M., Parmentier, E.M., Wysession, M.E., and Clarke, T.J. 2000. Shear wave splitting, continental keels, and patterns of mantle flow. *J. Geophys. Res.*, **105**, 6255–6276.

GSETT-3. 1995. *GSETT-3 Documentation, Conference Room Paper Series*. Tech. Rep. 243. United Nations, Comprehensive Test-ban Treaty Office.

Hamming, R.W. 1989. *Digital Filters*. Englewood Cliffs, NJ: Prentice-Hall.

Harris, D. 1990. *XAPiir: A Recursive Digital Filtering Package*. Tech. Rep. UCRL-ID-106005. Lawrence Livermore National Laboratory.

Helffrich, G. 2006. Extended-time multitaper frequency domain cross-correlation receiver-function estimation. *Bull. Seism. Soc. Am.*, **96**(1), 344–347.

IRIS. 2006. *SEED Reference Manual*, 3rd edn. Incorporated Research Institutions for Seismology.

Kennett, B.L.N., and Engdahl, E.R. 1991. Traveltimes for global earthquake and phase identification. *Geophys. J. Int.*, **105**, 429–465.

Kennett, B.L.N., Engdahl, E.R., and Buland, R. 1995. Constraints on seismic velocities in the Earth from traveltimes. *Geophys. J. Int.*, **122**, 108–124.

Kradolfer, U. 1996. AutoDRM: the first five years. *Seism. Res. Lett.*, **67**, 30–33.

Lacoss, R.T. 1971. Data adaptive spectral analysis methods. *Geophysics*, **36**(4), 661–675.

Langston, C.A. 1979. Structure under Mount Rainer, Washington, inferred from teleseismic body waves. *J. Geophys. Res.*, **84**, 4749–4762.

Ligorrìa, J., and Ammon, C.J. 1999. Iterative deconvolution and receiver-function estimation. *Bull. Seism. Soc. Am.*, **89**(5), 1395–1400.

McFadden, P.L., Drummond, B.J., and Kravis, S. 1986. The *N*th-root stack: theory, applications, and examples. *Geophysics*, **51**, 1879–1892.

Morelli, A., and Dziewonski, A.M. 1993. Body-wave traveltimes and a spherically symmetric P- and S-wave velocity model. *Geophys. J. Int.*, **112**, 178–194.

Obara, K., Kasahara, K., Hori, S., and Okada, Y. 2005. A densely distributed high-sensitivity seismograph network in Japan: Hi-net by National Research Institute for Earth Science and Disaster Prevention. *Rev. Sci. Instrum.*, **76**, doi:10.1063/1.1854197.

Park, J., and Levin, V. 2000. Receiver functions from multiple-taper spectral correlation estimates. *Bull. Seism. Soc. Am.*, **90**(6), 1507–1520.

Restivo, A., and Helffrich, G. 1999. Teleseismic shear-wave splitting measurement in noisy environments. *Geophys. J. Int.*, **137**, 821–830.

Rost, S., and Thomas, C. 2002. Array seismology: methods and applications. *Rev. Geophys.*, **40**, doi:10.1029/2000RG000100.

Savage, M.K. 1999. Seismic anisotropy and mantle deformation: what have we learned from shear wave splitting? *Rev. Geophys.*, **37**, 65–106.

Scargle, J.D. 1989. Studies in astronomical time series analysis. III. Fourier transforms, autocorrelation functions, and cross-correlation functions of unevenly spaced data. *Astrophys. J.*, **343**, 874–887.

Schimmel, M., and Paulssen, H. 1997. Noise reduction and detection of weak, coherent signals through phase-weighted stacks. *Geophys. J. Int.*, **130**, 497–505.

Silver, P., and Chan, G. 1991. Shear wave splitting and subcontinental mantle deformation. *J. Geophys. Res.*, **96**, 16429–16454.

Thompson, D.A., Bastow, I.D., Helffrich, G., Kendall, J.-M., Wookey, J., Snyder, D.B., and Eaton, D.W. 2010. Precambrian crustal evolution: seismic constraints from the Canadian Shield. *Earth Planet. Sci. Lett.*, **297**, 655–666.

Thomson, D.J. 1982. Spectrum estimation and harmonic analysis. *Proc. IEEE*, **70**(9), 1055–1096.

Verdon, J.P., Kendall, J.M., and Wuestefeld, A. 2009. Imaging fractures and sedimentary fabrics using shear wave splitting measurements made on passive seismic data. *Geophys. J. Int.*, **179**(2), 1245–1254.

Vergino, E., and Snoke, A. 1993. SAC and MAP. *IRIS Newsletter*, **XII**(2), 7–8.

Wiggins, R.A. 1976. Interpolation of digitized curves. *Bull. Seism. Soc. Am.*, **66**, 2077–2081.

Wookey, J., and Helffrich, G. 2008. Inner-core shear-wave anisotropy and texture from an observation of PKJKP waves. *Nature*, **454**(7206), 873–876.

Wuestefeld, A., and Bokelmann, G. 2007. Null detection in shear-wave splitting measurements. *Bull. Seism. Soc. Am.*, **97**(4), 1204–1211.

Zandt, G., Myers, S.C., and Wallace, T.C. 1995. Crust and mantle structure across the Basin and Range Colorado Plateau boundary at 47°N latitude and implications for Cenozoic extensional mechanism. *J. Geophys. Res.*, **100**(B6), 10529–10548.

Zhu, L., and Kanamori, H. 2000. Moho depth variation in southern California from teleseismic receiver functions. *J. Geophys. Res.*, **105**(B2), 2969–2980.

Index